

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

August 26, 2025

Abstract

This document is the documented code of the LaTeX package `nicematrix`. It is *not* its user's guide. The guide of utilisation is the document `nicematrix.pdf` (with a French traduction: `nicematrix-french.pdf`).

The development of the extension `nicematrix` is done on the following GitHub depot:
<https://github.com/fpantigny/nicematrix>

1 Declaration of the package and packages loaded

The prefix `nicematrix` has been registered for this package.

See: <http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf>
<@=@=nicematrix>

First, we load `pgfcore` and the module `shapes`. We do so because it's not possible to use `\usepgfmodule` in `\ExplSyntaxOn`.

```
1 \RequirePackage{pgfcore}
2 \usepgfmodule{shapes}
```

We give the traditional declaration of a package written with the L3 programming layer.

```
3 \ProvidesExplPackage
4   {nicematrix}
5   {\myfiledate}
6   {\myfileversion}
7   {Enhanced arrays with the help of PGF/TikZ}
8 \msg_new:nnn { nicematrix } { latex-too-old }
9   {
10    Your-LaTeX-release-is-too-old. \
11    You-need-at-least-a-the-version-of-2023-11-01
12   }
13 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
14 \IfFormatAtLeastTF
15   { 2023-11-01 }
16   { }
17   { \msg_fatal:nn { nicematrix } { latex-too-old } }
18 \ProvideDocumentCommand { \IfPackageLoadedT } { m m }
19   { \IfPackageLoadedTF { #1 } { #2 } { } }
20
21 \ProvideDocumentCommand { \IfPackageLoadedF } { m m }
22   { \IfPackageLoadedTF { #1 } { } { #2 } }
```

*This document corresponds to the version 7.2b of `nicematrix`, at the date of 2025/08/26.

The command for the treatment of the options of `\usepackage` is at the end of this package for technical reasons.

```
23 \RequirePackage { amsmath }
```

```
24 \RequirePackage { array }
```

In the version 2.6a of `array`, important modifications have been done for the Tagging Project.

```
25 \bool_const:Nn \c_@@_recent_array_bool
26 { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
27 \bool_const:Nn \c_@@_testphase_table_bool
28 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
```

```
29 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
30 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
31 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
32 \cs_generate_variant:Nn \@@_error:n { n e }
33 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
34 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
35 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
36 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
```

With Overleaf (and also in TeXPage), by default, a document is compiled in non-stop mode. When there is an error, there is no way to the user to use the key `H` in order to have more information. That's why we decide to put that piece of information (for the messages with such information) in the main part of the message when the key `messages-for-Overleaf` is used (at load-time).

```
37 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
38 {
39   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
40     { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
41     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
42 }
```

We also create a command which will generate usually an error but only a warning on Overleaf. The argument is given by curryfication.

```
43 \cs_new_protected:Npn \@@_error_or_warning:n
44 {
45   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
46     { \@@_warning:n }
47     { \@@_error:n }
48 }
```

We try to detect whether the compilation is done on Overleaf. We use `\c_sys_jobname_str` because, with Overleaf, the value of `\c_sys_jobname_str` is always "output".

```
49 \bool_new:N \g_@@_messages_for_Overleaf_bool
50 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
51 {
52   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
53   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
54 }
```

```
55 \@@_msg_new:nn { mdwtab-loaded }
56 {
57   The-packages-'mdwtab'-and-'nicematrix'-are-incompatible.~
58   This-error-is-fatal.
59 }
```

```
60 \hook_gput_code:nnn { begindocument / end } { . }
61 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab-loaded } } }
```

2 Collecting options

The following technique allows to create user commands with the ability to put an arbitrary number of `[list of (key=val)]` after the name of the command.

Example :

```
\@@_collect_options:n { \F } [x=a,y=b] [z=c,t=d] { arg }  
will be transformed in : \F{x=a,y=b,z=c,t=d}{arg}
```

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,
the command `\G` takes in an arbitrary number of optional arguments between square brackets.
Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```
62 \cs_new_protected:Npn \@@_collect_options:n #1  
63 {  
64   \peek_meaning:NTF [  
65     { \@@_collect_options:nw { #1 } }  
66     { #1 { } }  
67   }  
}
```

We use `\NewDocumentCommand` in order to be able to allow nested brackets within the argument between `[` and `]`.

```
68 \NewDocumentCommand \@@_collect_options:nw { m r[] }  
69 { \@@_collect_options:nn { #1 } { #2 } }  
70  
71 \cs_new_protected:Npn \@@_collect_options:nn #1 #2  
72 {  
73   \peek_meaning:NTF [  
74     { \@@_collect_options:nw { #1 } { #2 } }  
75     { #1 { #2 } }  
76   }  
77  
78 \cs_new_protected:Npn \@@_collect_options:nw #1#2[#3]  
79 { \@@_collect_options:nn { #1 } { #2 , #3 } }
```

3 Technical definitions

The following constants are defined only for efficiency in the tests.

```
80 \tl_const:Nn \c_@@_b_tl { b }  
81 \tl_const:Nn \c_@@_c_tl { c }  
82 \tl_const:Nn \c_@@_l_tl { l }  
83 \tl_const:Nn \c_@@_r_tl { r }  
84 \tl_const:Nn \c_@@_all_tl { all }  
85 \tl_const:Nn \c_@@_dot_tl { . }  
86 \str_const:Nn \c_@@_r_str { r }  
87 \str_const:Nn \c_@@_c_str { c }  
88 \str_const:Nn \c_@@_l_str { l }
```

The following token list will be used for definitions of user commands (with `\NewDocumentCommand`) with an embellishment using an *underscore* (there may be problems because of the catcode of the underscore).

```
89 \tl_new:N \l_@@_argspec_tl
```

```

90 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
91 \cs_generate_variant:Nn \str_set:Nn { N o }
92 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
93 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
94 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
95 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
96 \cs_generate_variant:Nn \dim_min:nn { v }
97 \cs_generate_variant:Nn \dim_max:nn { v }

98 \hook_gput_code:nnn { begindocument } { . }
99 {
100   \IfPackageLoadedTF { tikz }
101   {

```

In some constructions, we will have to use a `{pgfpicture}` which *must* be replaced by a `{tikzpicture}` if Tikz is loaded. However, this switch between `{pgfpicture}` and `{tikzpicture}` can't be done dynamically with a conditional because, when the Tikz library `external` is loaded by the user, the pair `\tikzpicture-\endtikzpicture` (or `\begin{tikzpicture}-\end{tikzpicture}`) must be statically “visible” (even when externalization is not activated).

That's why we create `\c_@@_pgfortikzpicture_tl` and `\c_@@_endpgfortikzpicture_tl` which will be used to construct in a `\hook_gput_code:nnn { begindocument } { . }` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

102     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
103     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
104   }
105   {
106     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
107     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
108   }
109 }

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date April 2025, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

110 \IfClassLoadedTF { revtex4-1 }
111 { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
112 {
113   \IfClassLoadedTF { revtex4-2 }
114   { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
115   {

```

Maybe one of the previous classes will be loaded inside another class... We try to detect that situation.

```

116     \cs_if_exist:NT \rvtx@ifformat@geq
117     { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
118     { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
119   }
120 }

```

If the final user uses `nicematrix`, PGF/Tikz will write instruction `\pgfsyspdfmark` in the aux file. If he changes its mind and no longer loads `nicematrix`, an error may occur at the next compilation because of remanent instructions `\pgfsyspdfmark` in the aux file. With the following code, we try to avoid that situation.

```

121 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
122 {
123   \iow_now:Nn \@mainaux
124   {
125     \ExplSyntaxOn
126     \cs_if_free:NT \pgfsyspdfmark
127     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
128     \ExplSyntaxOff
129   }
130   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
131 }

```

We define a command `\iddots` similar to `\ddots` (`\ddots`) but with dots going forward (`\iddots`). We use `\ProvideDocumentCommand` and so, if the command `\iddots` has already been defined (for example by the package `mathdots`), we don't define it again.

```

132 \ProvideDocumentCommand \iddots { }
133 {
134   \mathinner
135     {
136       \mkern 1 mu
137       \box_move_up:nn { 1 pt } { \hbox { . } }
138       \mkern 2 mu
139       \box_move_up:nn { 4 pt } { \hbox { . } }
140       \mkern 2 mu
141       \box_move_up:nn { 7 pt }
142         { \vbox:n { \kern 7 pt \hbox { . } } }
143       \mkern 1 mu
144     }
145 }

```

This definition is a variant of the standard definition of `\ddots`.

In the `aux` file, we will have the references of the PGF/Tikz nodes created by `nicematrix`. However, when `booktabs` is used, some nodes (more precisely, some `row` nodes) will be defined twice because their position will be modified. In order to avoid an error message in this case, we will redefine `\pgfutil@check@rerun` in the `aux` file.

```

146 \hook_gput_code:nnn { begindocument } { . }
147 {
148   \IfPackageLoadedT { booktabs }
149     { \iow_now:Nn \@mainaux { \nicematrix@redefine@check@rerun } }
150 }
151 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
152 {
153   \let @@_old_pgfutil@check@rerun \pgfutil@check@rerun

```

The new version of `\pgfutil@check@rerun` will not check the PGF nodes whose names start with `nm-` (which is the prefix for the nodes created by `nicematrix`).

```

154   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
155     {

```

`\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

156       \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
157       { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
158     }
159 }

```

We have to know whether `colortbl` is loaded in particular for the redefinition of `\everycr`.

```

160 \hook_gput_code:nnn { begindocument } { . }
161 {
162   \cs_set_protected:Npe \@@_everycr:
163     {
164       \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
165       { \noalign { \@@_in_everycr: } }
166     }
167   \IfPackageLoadedTF { colortbl }
168     {
169       \cs_set_eq:NN \@@_old_cellcolor: \cellcolor
170       \cs_set_eq:NN \@@_old_rowcolor: \rowcolor
171       \cs_new_protected:Npn \@@_revert_colortbl:
172         {
173           \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
174           {
175             \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
176             \cs_set_eq:NN \rowcolor \@@_old_rowcolor:

```

```

177     }
178 }

```

When `colortbl` is used, we have to catch the tokens `\columncolor` in the preamble because, otherwise, `colortbl` will catch them and the colored panels won't be drawn by `nicematrix` but by `colortbl` (with an output which is not perfect).

```

179 \cs_new_protected:Npn \@@_replace_columncolor:
180 {
181   \tl_replace_all:Nnn \g_@@_array_preamble_tl
182   { \columncolor }
183   { \@@_columncolor_preamble }

```

`\@@_column_preamble`, despite its name, will be defined with `\NewDocumentCommand` because it takes in an optional argument between square brackets in first position for the colorimetric space.

```

184   }
185 }
186 {
187   \cs_new_protected:Npn \@@_revert_colortbl: { }
188   \cs_new_protected:Npn \@@_replace_columncolor:
189   { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if `colortbl` is not loaded.

```

190 \def \CT@arc@ { }
191 \def \arrayrulecolor #1 # { \CT@arc@ { #1 } }
192 \def \CT@arc@ #1 #2
193 {
194   \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
195   { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
196 }

```

Idem for `\CT@drs@`.

```

197 \def \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
198 \def \CT@drs@ #1 #2
199 {
200   \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
201   { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
202 }
203 \def \hline
204 {
205   \noalign { \ifnum 0 = ` } \fi
206   \cs_set_eq:NN \hskip \vskip
207   \cs_set_eq:NN \vrule \hrule
208   \cs_set_eq:NN \@width \@height
209   { \CT@arc@ \vline }
210   \futurelet \reserved@a
211   \@xhline
212 }
213 }
214 }

```

We have to redefine `\cline` for several reasons. The command `\@@_cline:` will be linked to `\cline` in the beginning of `{NiceArrayWithDelims}`. The following commands must *not* be protected.

```

215 \cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
216 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
217 {
218   \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
219   \int_compare:nNnT { #1 } > { \c_one_int }
220   { \multispan { \int_eval:n { #1 - 1 } } & }
221   \multispan { \int_eval:n { #2 - #1 + 1 } }
222   {
223     \CT@arc@
224     \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`¹

```
225     \skip_horizontal:N \c_zero_dim
226   }
```

Our `\everycr` has been modified. In particular, the creation of the row node is in the `\everycr` (maybe we should put it with the incrementation of `\c@iRow`). Since the following `\cr` correspond to a “false row”, we have to nullify `\everycr`.

```
227   \everycr { }
228   \cr
229   \noalign { \skip_vertical:n { - \arrayrulewidth } }
230 }
```

The following version of `\cline` spreads the array of a quantity equal to `\arrayrulewidth` as does `\hline`. It will be loaded excepted if the key `standard-cline` has been used.

```
231 \cs_set:Npn \@@_cline:
```

We have to act in a fully expandable way since there may be `\noalign` (in the `\multispan`) to detect. That’s why we use `\@@_cline_i:en`.

```
232 { \@@_cline_i:en \l_@@_first_col_int }
```

The command `\cline_i:nn` has two arguments. The first is the number of the current column (it *must* be used in that column). The second is a standard argument of `\cline` of the form *i-j* or the form *i*.

```
233 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
234 \cs_generate_variant:Nn \@@_cline_i:nn { e }
235 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
236 {
237   \tl_if_empty:nTF { #3 }
238     { \@@_cline_iii:w #1|#2-#2 \q_stop }
239     { \@@_cline_ii:w #1|#2-#3 \q_stop }
240 }
241 \cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
242 { \@@_cline_iii:w #1|#2-#3 \q_stop }
243 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
244 {
```

Now, `#1` is the number of the current column and we have to draw a line from the column `#2` to the column `#3` (both included).

```
245   \int_compare:nNnT { #1 } < { #2 }
246     { \multispan { \int_eval:n { #2 - #1 } } & }
247   \multispan { \int_eval:n { #3 - #2 + 1 } }
248     {
249       \CT@arc@
250       \leaders \hrule \@height \arrayrulewidth \hfill
251       \skip_horizontal:N \c_zero_dim
252     }
```

You look whether there is another `\cline` to draw (the final user may put several `\cline`).

```
253   \peek_meaning_remove_ignore_spaces:NTF \cline
254     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
255     { \everycr { } \cr }
256 }
```

The following command will be nullified in the environment `{NiceTabular}`, `{NiceTabular*}` and `{NiceTabularX}`.

```
257 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token
```

¹See question 99041 on TeX StackExchange.

```

258 \cs_new_protected:Npn \@@_set_CTarc:n #1
259 {
260   \tl_if_blank:nF { #1 }
261   {
262     \tl_if_head_eq_meaning:nNTF { #1 } [
263       { \def \CT@arc@ { \color #1 } }
264       { \def \CT@arc@ { \color { #1 } } }
265     ]
266   }
267 \cs_generate_variant:Nn \@@_set_CTarc:n { o }

```

```

268 \cs_new_protected:Npn \@@_set_CTdrsc:n #1
269 {
270   \tl_if_head_eq_meaning:nNTF { #1 } [
271     { \def \CT@drsc@ { \color #1 } }
272     { \def \CT@drsc@ { \color { #1 } } }
273   ]
274 \cs_generate_variant:Nn \@@_set_CTdrsc:n { o }

```

The following command must *not* be protected since it will be used to write instructions in the `\g_@@_pre_code_before_tl`.

```

275 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
276 {
277   \tl_if_head_eq_meaning:nNTF { #2 } [
278     { #1 #2 }
279     { #1 { #2 } }
280   ]
281 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

The following command must be protected because of its use of the command `\color`.

```

282 \cs_new_protected:Npn \@@_color:n #1
283 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
284 \cs_generate_variant:Nn \@@_color:n { o }

```

```

285 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
286 {
287   \tl_set_rescan:Nno
288     #1
289     {
290       \char_set_catcode_other:N >
291       \char_set_catcode_other:N <
292     }
293   #1
294 }

```

The L3 programming layer provides scratch dimensions `\l_tmpa_dim` and `\l_tmpb_dim`. We create several more in the same spirit.

```

295 \dim_new:N \l_@@_tmpc_dim
296 \dim_new:N \l_@@_tmpd_dim
297 \dim_new:N \l_@@_tmpe_dim
298 \dim_new:N \l_@@_tmpf_dim

299 \tl_new:N \l_@@_tmpc_tl
300 \tl_new:N \l_@@_tmpd_tl

301 \int_new:N \l_@@_tmpc_int

```


4 Parameters

The following counter will count the environments `{NiceArray}`. The value of this counter will be used to prefix the names of the Tikz nodes created in the array.

```
302 \int_new:N \g_@@_env_int
```

The following command is only a syntactic shortcut. It must *not* be protected (it will be used in names of PGF nodes).

```
303 \cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }
```

The command `\NiceMatrixLastEnv` is not used by the package `nicematrix`. It's only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it's meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name). This command *must* be expandable since it will be used in `pgf` nodes.

```
304 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
305 { \int_use:N \g_@@_env_int }
```

The following command is only a syntactic shortcut. The `q` in `qpoint` means *quick*.

```
306 \cs_new_protected:Npn \@@_qpoint:n #1
307 { \pgfpointanchor { \@@_env: - #1 } { center } }
```

If the user uses `{NiceTabular}`, `{NiceTabular*}` or `{NiceTabularX}`, we will raise the following flag.

```
308 \bool_new:N \l_@@_tabular_bool
```

`\g_@@_delims_bool` will be true for the environments with delimiters (ex. : `{pNiceMatrix}`, `{pNiceArray}`, `\pAutoNiceMatrix`, etc.).

```
309 \bool_new:N \g_@@_delims_bool
310 \bool_gset_true:N \g_@@_delims_bool
```

In fact, if there is delimiters in the preamble of `{NiceArray}` (eg: `[cccc]`), this boolean will be set to false.

The following boolean will be equal to `true` in the environments which have a preamble (provided by the final user): `{NiceTabular}`, `{NiceArray}`, `{pNiceArray}`, etc.

```
311 \bool_new:N \l_@@_preamble_bool
312 \bool_set_true:N \l_@@_preamble_bool
```

We need a special treatment for `{NiceMatrix}` when `vlines` is not used, in order to retrieve `\arraycolsep` on both sides.

```
313 \bool_new:N \l_@@_NiceMatrix_without_vlines_bool
```

The following counter will count the environments `{NiceMatrixBlock}`.

```
314 \int_new:N \g_@@_NiceMatrixBlock_int
```

It's possible to put tabular notes (with `\tabularnote`) in the caption if that caption is composed *above* the tabular. In such case, we will count in `\g_@@_notes_caption_int` the number of uses of the command `\tabularnote` *without optional argument* in that caption.

```
315 \int_new:N \g_@@_notes_caption_int
```

The dimension `\l_@@_columns_width_dim` will be used when the options specify that all the columns must have the same width (but, if the key `columns-width` is used with the special value `auto`, the boolean `\l_@@_auto_columns_width_bool` also will be raised).

```
316 \dim_new:N \l_@@_columns_width_dim
```

The dimension `\l_@@_col_width_dim` will be available in each cell which belongs to a column of fixed width: `w{...}{...}`, `W{...}{...}`, `p{...}`, `m{...}`, `b{...}` but also `X` (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands `\Block`. A non positive value means that the column has no fixed width (it's a column of type `c`, `r`, `l`, etc.).

```
317 \dim_new:N \l_@@_col_width_dim
318 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

The following counters will be used to count the numbers of rows and columns of the array.

```
319 \int_new:N \g_@@_row_total_int
320 \int_new:N \g_@@_col_total_int
```

The following parameter will be used by `\@@_create_row_node`: to avoid to create the same row-node twice (at the end of the array).

```
321 \int_new:N \g_@@_last_row_node_int
```

The following counter corresponds to the key `nb-rows` of the command `\RowStyle`.

```
322 \int_new:N \l_@@_key_nb_rows_int
```

The following token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are `r`, `l`, `c` and `j`. For example, a column `p[1]{3cm}` will provide the value `l` for all the cells of the column.

```
323 \tl_new:N \l_@@_hpos_cell_tl
324 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

When there is a mono-column block (created by the command `\Block`), we want to take into account the width of that block for the width of the column. That's why we compute the width of that block in the `\g_@@_blocks_wd_dim` and, after the construction of the box `\l_@@_cell_box`, we change the width of that box to take into account the length `\g_@@_blocks_wd_dim`.

```
325 \dim_new:N \g_@@_blocks_wd_dim
```

Idem for the mono-row blocks.

```
326 \dim_new:N \g_@@_blocks_ht_dim
327 \dim_new:N \g_@@_blocks_dp_dim
```

The following dimension correspond to the key `width` (which may be fixed in `\NiceMatrixOptions` but also in an environment `{NiceTabular}`).

```
328 \dim_new:N \l_@@_width_dim
```

The clist `\g_@@_names_clist` will be the list of all the names of environments used (via the option `name`) in the document: two environments must not have the same name. However, it's possible to use the option `allow-duplicate-names`.

```
329 \clist_new:N \g_@@_names_clist
```

We want to know whether we are in an environment of `nicematrix` because we will raise an error if the user tries to use nested environments.

```
330 \bool_new:N \l_@@_in_env_bool
```

The following key corresponds to the key `notes/detect_duplicates`.

```
331 \bool_new:N \l_@@_notes_detect_duplicates_bool
332 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

```
333 \bool_new:N \l_@@_initial_open_bool
334 \bool_new:N \l_@@_final_open_bool
335 \bool_new:N \l_@@_Vbrace_bool
```

If the user uses `{NiceTabular*}`, the width of the tabular (in the first argument of the environment `{NiceTabular*}`) will be stored in the following dimension.

```
336 \dim_new:N \l_@@_tabular_width_dim
```

The following dimension will be used for the total width of composite rules (*total* means that the spaces on both sides are included).

```
337 \dim_new:N \l_@@_rule_width_dim
```

The key `color` in a command of rule such as `\Hline` (or the specifier “|” in the preamble of an environment).

```
338 \tl_new:N \l_@@_rule_color_tl
```

The following boolean will be raised when the command `\rotate` is used.

```
339 \bool_new:N \g_@@_rotate_bool
```

The following boolean will be raised when the command `\rotate` is used with the key `c`.

```
340 \bool_new:N \g_@@_rotate_c_bool
```

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag (the `X` columns of `nicematrix` are inspired by those of `tabularx`). You will use that flag for the blocks.

```
341 \bool_new:N \l_@@_X_bool
```

`\l_@@_V_of_X_bool` during the construction of the preamble when a column of type `X` uses the key `V` (whose name is inspired by the columns `V` of the extension `varwidth`).

```
342 \bool_new:N \l_@@_V_of_X_bool
```

The flag `g_@@_V_of_X_bool` will be raised when there is at least in the tabular a column of type `X` using the key `V`.

```
343 \bool_new:N \g_@@_V_of_X_bool
```

```
344 \bool_new:N \g_@@_caption_finished_bool
```

The following boolean will be raised when the key `no-cell-nodes` is used.

```
345 \bool_new:N \l_@@_no_cell_nodes_bool
```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }`).

```
346 \tl_new:N \g_@@_aux_tl
```

During the second run, if information concerning the current environment has been found in the `aux` file, the following flag will be raised.

```
347 \bool_new:N \g_@@_aux_found_bool
```

In particular, in that `aux` file, there will be, for each environment of `nicematrix`, an affectation for the the following sequence that will contain information about the size of the array.

```
348 \seq_new:N \g_@@_size_seq
```

```
349 \tl_new:N \g_@@_left_delim_tl
```

```
350 \tl_new:N \g_@@_right_delim_tl
```

The token list `\g_@@_user_preamble_tl` will contain the preamble provided by the the final user of `nicematrix` (eg the preamble of an environment `{NiceTabular}`).

```
351 \tl_new:N \g_@@_user_preamble_tl
```

The token list `\g_@@_array_preamble_tl` will contain the preamble constructed by `nicematrix` for the environment `{array}` (of `array`).

```
352 \tl_new:N \g_@@_array_preamble_tl
```

For `\multicolumn`.

```
353 \tl_new:N \g_@@_preamble_tl
```

The following parameter corresponds to the key `columns-type` of the environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. and also the key `matrix / columns-type` of `\NiceMatrixOptions`.

```
354 \tl_new:N \l_@@_columns_type_tl
355 \str_set:Nn \l_@@_columns_type_tl { c }
```

The following parameters correspond to the keys `down`, `up` and `middle` of a command such as `\Cdots`. Usually, the final user doesn't use that keys directly because he uses the syntax with the embellishments `_`, `^` and `:`.

```
356 \tl_new:N \l_@@_xdots_down_tl
357 \tl_new:N \l_@@_xdots_up_tl
358 \tl_new:N \l_@@_xdots_middle_tl
```

We will store in the following sequence information provided by the instructions `\rowlistcolors` in the main array (not in the `\CodeBefore`).

```
359 \seq_new:N \g_@@_rowlistcolors_seq
```

```
360 \cs_new_protected:Npn \@@_test_if_math_mode:
361 {
362   \if_mode_math: \else:
363     \@@_fatal:n { Outside~math~mode }
364   \fi:
365 }
```

The list of the columns where vertical lines in sub-matrices (`vlism`) must be drawn. Of course, the actual value of this sequence will be known after the analyse of the preamble of the array.

```
366 \seq_new:N \g_@@_cols_vlism_seq
```

The following colors will be used to memorize the color of the potential “first col” and the potential “first row”.

```
367 \colorlet { nicematrix-last-col } { . }
368 \colorlet { nicematrix-last-row } { . }
```

The following string is the name of the current environment or the current command of `nicematrix` (despite its name which contains `env`).

```
369 \str_new:N \g_@@_name_env_str
```

The following string will contain the word *command* or *environment* whether we are in a command of `nicematrix` or in an environment of `nicematrix`. The default value is *environment*.

```
370 \tl_new:N \g_@@_com_or_env_str
371 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
372 \bool_new:N \l_@@_bold_row_style_bool
```

The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains `env`). This command must *not* be protected since it will be used in error messages and we have to use `\str_if_eq:eeTF` and not `\tl_if_eq:eeTF` because we need to be fully expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
373 \cs_new:Npn \@@_full_name_env:
374 {
375   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
376     { command \space \c_backslash_str \g_@@_name_env_str }
377     { environment \space \{ \g_@@_name_env_str \} }
378 }
```

```
379 \tl_new:N \g_@@_cell_after_hook_tl % 2025/03/22
```

For the key code of the command `\SubMatrix` (itself in the main `\CodeAfter`), we will use the following token list.

```
380 \tl_new:N \l_@@_code_tl
```

For the key `pgf-node-code`. That code will be used when the nodes of the cells (that is to say the nodes of the form `i-j`) will be created.

```
381 \tl_new:N \l_@@_pgf_node_code_tl
```

The so-called `\CodeBefore` is split in two parts because we want to control the order of execution of some instructions.

```
382 \tl_new:N \g_@@_pre_code_before_tl
383 \tl_new:N \g_nicematrix_code_before_tl
```

The value of the key `code-before` will be added to the left of `\g_@@_pre_code_before_tl`. Idem for the code between `\CodeBefore` and `\Body`.

The so-called `\CodeAfter` is split in two parts because we want to control the order of execution of some instructions.

```
384 \tl_new:N \g_@@_pre_code_after_tl
385 \tl_new:N \g_nicematrix_code_after_tl
```

The `\CodeAfter` provided by the final user (with the key `code-after` or the keyword `\CodeAfter`) will be stored in the second token list.

```
386 \bool_new:N \l_@@_in_code_after_bool
```

The following parameter will be raised when a block contains an ampersand (`&`) in its content (`=label`).

```
387 \bool_new:N \l_@@_ampersand_bool
```

The counters `\l_@@_old_iRow_int` and `\l_@@_old_jCol_int` will be used to save the values of the potential LaTeX counters `iRow` and `jCol`. These LaTeX counters will be restored at the end of the environment.

```
388 \int_new:N \l_@@_old_iRow_int
389 \int_new:N \l_@@_old_jCol_int
```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

The following sequence will contain the names (without backslash) of the commands created by `custom-line` by the key `command` or `ccommand` (commands used by the final user in order to draw horizontal rules).

```
390 \seq_new:N \l_@@_custom_line_commands_seq
```

The following token list corresponds to the key `rules/color` available in the environments.

```
391 \tl_new:N \l_@@_rules_color_tl
```

The sum of the weights of all the X-columns in the preamble.

```
392 \fp_new:N \g_@@_total_X_weight_fp
```

If there is at least one X-column in the preamble of the array, the following flag will be raised via the `aux` file. The length `\l_@@_x_columns_dim` will be the width of X-columns of weight 1.0 (the width of a column of weight x will be that dimension multiplied by x). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
393 \bool_new:N \l_@@_X_columns_aux_bool
394 \dim_new:N \l_@@_X_columns_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
395 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the `col` nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of `col` nodes).

```
396 \bool_new:N \g_@@_row_of_col_done_bool
```

It's possible to use the command `\NotEmpty` to specify explicitly that a cell must be considered as non empty by `nicematrix` (the Tikz nodes are constructed only in the non empty cells).

```
397 \bool_new:N \g_@@_not_empty_cell_bool
```

```
398 \tl_new:N \l_@@_code_before_tl
```

```
399 \bool_new:N \l_@@_code_before_bool
```

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

```
400 \tl_new:N \g_@@_row_style_tl
```

The following dimensions will be used when drawing the dotted lines.

```
401 \dim_new:N \l_@@_x_initial_dim
```

```
402 \dim_new:N \l_@@_y_initial_dim
```

```
403 \dim_new:N \l_@@_x_final_dim
```

```
404 \dim_new:N \l_@@_y_final_dim
```

```
405 \dim_new:N \g_@@_dp_row_zero_dim
```

```
406 \dim_new:N \g_@@_ht_row_zero_dim
```

```
407 \dim_new:N \g_@@_ht_row_one_dim
```

```
408 \dim_new:N \g_@@_dp_ante_last_row_dim
```

```
409 \dim_new:N \g_@@_ht_last_row_dim
```

```
410 \dim_new:N \g_@@_dp_last_row_dim
```

Some cells will be declared as “empty” (for example a cell with an instruction `\Cdots`).

```
411 \bool_new:N \g_@@_empty_cell_bool
```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

```
412 \dim_new:N \g_@@_width_last_col_dim
```

```
413 \dim_new:N \g_@@_width_first_col_dim
```

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces: `{imin}{jmin}{imax}{jmax}{options}{contents}`.

The variable is global because it will be modified in the cells of the array.

```
414 \seq_new:N \g_@@_blocks_seq
```

We also manage a sequence of the *positions* of the blocks. In that sequence, each block is represented by only five components: `{imin}{jmin}{imax}{jmax}{name}`. A block with the key `hvlines` won't appear in that sequence (otherwise, the lines in that block would not be drawn!).

```
415 \seq_new:N \g_@@_pos_of_blocks_seq
```

In fact, this sequence will also contain the positions of the cells with a `\diagbox`. The sequence `\g_@@_pos_of_blocks_seq` will be used when we will draw the rules (which respect the blocks).

In the `\CodeBefore`, the value of `\g_@@_pos_of_blocks_seq` will be the value read in the `aux` file from a previous run. However, in the `\CodeBefore`, the commands `\EmptyColumn` and `\EmptyRow` will write virtual positions of blocks in the following sequence.

```
416 \seq_new:N \g_@@_future_pos_of_blocks_seq
```

The, after the execution of the `\CodeBefore`, the sequence `\g_@@_pos_of_blocs_seq` will be erased and replaced by the value of `\g_@@_future_pos_of_blocks_seq`.

We will also manage a sequence for the positions of the dotted lines. These dotted lines are created in the array by `\Cdots`, `\Vdots`, `\Ddots`, etc. However, their positions, that is to say, their extremities, will be determined only after the construction of the array. In this sequence, each item contains five components: `{imin}{jmin}{imax}{jmax}{ name}`.

```
417 \seq_new:N \g_@@_pos_of_xdots_seq
```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won't be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That's why we keep the information of all that stroken blocks in the following sequence.

```
418 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
419 \clist_new:N \l_@@_corners_cells_clist
```

The list of the names of the potential `\SubMatrix` in the `\CodeAfter` of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given `\SubMatrix`).

```
420 \seq_new:N \g_@@_submatrix_names_seq
```

The following flag will be raised if the key `width` is used in an environment `{NiceTabular}` (not in a command `\NiceMatrixOptions`). You use it to raise an error when this key is used while no column `X` is used.

```
421 \bool_new:N \l_@@_width_used_bool
```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with $n > 1$ is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of n) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
422 \seq_new:N \g_@@_multicolumn_cells_seq
```

```
423 \seq_new:N \g_@@_multicolumn_sizes_seq
```

By default, the diagonal lines will be parallelized². There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```
424 \int_new:N \g_@@_ddots_int
```

```
425 \int_new:N \g_@@_iddots_int
```

²It's possible to use the option `parallelize-diags` to disable this parallelization.

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```
426 \dim_new:N \g_@@_delta_x_one_dim
427 \dim_new:N \g_@@_delta_y_one_dim
428 \dim_new:N \g_@@_delta_x_two_dim
429 \dim_new:N \g_@@_delta_y_two_dim
```

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the code-before).

```
430 \int_new:N \l_@@_row_min_int
431 \int_new:N \l_@@_row_max_int
432 \int_new:N \l_@@_col_min_int
433 \int_new:N \l_@@_col_max_int

434 \int_new:N \l_@@_initial_i_int
435 \int_new:N \l_@@_initial_j_int
436 \int_new:N \l_@@_final_i_int
437 \int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
438 \int_new:N \l_@@_start_int
439 \int_set_eq:NN \l_@@_start_int \c_one_int
440 \int_new:N \l_@@_end_int
441 \int_new:N \l_@@_local_start_int
442 \int_new:N \l_@@_local_end_int
```

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `\CodeBefore`. Each sub-matrix is represented by an “object” of the form `{i}{j}{k}{l}` where i and j are the number of row and column of the upper-left cell and k and l the number of row and column of the lower-right cell.

```
443 \seq_new:N \g_@@_submatrix_seq
```

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble of course and without the potential exterior columns).

```
444 \int_new:N \g_@@_static_num_of_col_int
```

The following parameters correspond to the keys `fill`, `opacity`, `draw`, `tikz`, `borders`, and `rounded-corners` of the command `\Block`.

```
445 \tl_new:N \l_@@_fill_tl
446 \tl_new:N \l_@@_opacity_tl
447 \tl_new:N \l_@@_draw_tl
448 \seq_new:N \l_@@_tikz_seq
449 \clist_new:N \l_@@_borders_clist
450 \dim_new:N \l_@@_rounded_corners_dim
```

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by `nicematrix` when the key `corners` is used).

The following dimension corresponds to the key `rounded-corners` available in an individual environment `{NiceTabular}`. When that key is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```
451 \dim_new:N \l_@@_tab_rounded_corners_dim
```

The following token list correspond to the key `color` of the command `\Block` and also the key `color` of the command `\RowStyle`.

```
452 \tl_new:N \l_@@_color_tl
```


In the key `tikz` of a command `\Block` or in the argument of a command `\TikzEveryCell`, the final user puts a list of `tikz` keys. But, you have added another key, named `offset` (which means that an offset will be used for the frame of the block or the cell). The following parameter corresponds to that key.

```
453 \dim_new:N \l_@@_offset_dim
```

Here is the dimension for the width of the rule when a block (created by `\Block`) is stroked or when the key `hvlines` is used.

```
454 \dim_new:N \l_@@_line_width_dim
```

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

```
455 \str_new:N \l_@@_hpos_block_str
456 \str_set:Nn \l_@@_hpos_block_str { c }
457 \bool_new:N \l_@@_hpos_of_block_cap_bool
458 \bool_new:N \l_@@_p_block_bool
```

If the final user has used the special color “`nocolor`”, the following flag will be raised.

```
459 \bool_new:N \l_@@_nocolor_used_bool
```

For the vertical position, the possible values are `c`, `t`, `b`, `T` and `B` (but `\l_@@_vpos_block_str` will remain empty if the user doesn’t use a key for the vertical position).

```
460 \str_new:N \l_@@_vpos_block_str
```

Used when the key `draw-first` is used for `\Ddots` or `\Iddots`.

```
461 \bool_new:N \l_@@_draw_first_bool
```

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

```
462 \bool_new:N \l_@@_vlines_block_bool
463 \bool_new:N \l_@@_hlines_block_bool
```

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

```
464 \int_new:N \g_@@_block_box_int

465 \dim_new:N \l_@@_submatrix_extra_height_dim
466 \dim_new:N \l_@@_submatrix_left_xshift_dim
467 \dim_new:N \l_@@_submatrix_right_xshift_dim
468 \clist_new:N \l_@@_hlines_clist
469 \clist_new:N \l_@@_vlines_clist
470 \clist_new:N \l_@@_submatrix_hlines_clist
471 \clist_new:N \l_@@_submatrix_vlines_clist
```

The following key is set when the keys `hvlines` and `hvlines-except-borders` are used. It’s used only to change slightly the clipping path set by the key `rounded-corners` (for a `{tabular}`).

```
472 \bool_new:N \l_@@_hvlines_bool
```

The following flag will be used by (for instance) `\@@_vline_ii:`. When `\l_@@_dotted_bool` is true, a dotted line (with our system) will be drawn.

```
473 \bool_new:N \l_@@_dotted_bool
```

The following flag will be set to true during the composition of a caption specified (by the key `caption`).

```
474 \bool_new:N \l_@@_in_caption_bool
```

Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

- **First row**

The integer `\l_@@_first_row_int` is the number of the first row of the array. The default value is 1, but, if the option `first-row` is used, the value will be 0.

```
475 \int_new:N \l_@@_first_row_int
476 \int_set_eq:NN \l_@@_first_row_int \c_one_int
```

- **First column**

The integer `\l_@@_first_col_int` is the number of the first column of the array. The default value is 1, but, if the option `first-col` is used, the value will be 0.

```
477 \int_new:N \l_@@_first_col_int
478 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

- **Last row**

The counter `\l_@@_last_row_int` is the number of the potential “last row”, as specified by the key `last-row`. A value of `-2` means that there is no “last row”. A value of `-1` means that there is a “last row” but we don’t know the number of that row (the key `last-row` has been used without value and the actual value has not still been read in the `aux` file).

```
479 \int_new:N \l_@@_last_row_int
480 \int_set:Nn \l_@@_last_row_int { -2 }
```

If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.³

```
481 \bool_new:N \l_@@_last_row_without_value_bool
```

Idem for `\l_@@_last_col_without_value_bool`

```
482 \bool_new:N \l_@@_last_col_without_value_bool
```

- **Last column**

For the potential “last column”, we use an integer. A value of `-2` means that there is no last column. A value of `-1` means that we are in an environment without preamble (e.g. `{bNiceMatrix}`) and there is a last column but we don’t know its value because the user has used the option `last-col` without value. A value of `0` means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`): in this case, the key was necessary without argument. The command `\NiceMatrixOptions` also sets `\l_@@_last_col_int` to 0.

```
483 \int_new:N \l_@@_last_col_int
484 \int_set:Nn \l_@@_last_col_int { -2 }
```

However, we have also a boolean. Consider the following code:

³We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.

```

\begin{pNiceArray}{cc}[last-col]
1 & 2 \\
3 & 4
\end{pNiceArray}

```

In such a code, the “last column” specified by the key `last-col` is not used. We want to be able to detect such a situation and we create a boolean for that job.

```
485 \bool_new:N \g_@@_last_col_found_bool
```

This boolean is set to `false` at the end of `\@@_pre_array_ii:`.

In the last column, we will raise the following flag (it will be used by `\OnlyMainNiceMatrix`).

```
486 \bool_new:N \l_@@_in_last_col_bool
```

Some utilities

```
487 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
488 {
```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
489 \def \l_tmpa_tl { #1 }
490 \def \l_tmpb_tl { #2 }
491 }
```

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers.

```
492 \cs_new_protected:Npn \@@_expand_clist:N #1
493 {
494 \clist_if_in:NnF #1 { all }
495 {
496 \clist_clear:N \l_tmpa_clist
497 \clist_map_inline:Nn #1
498 {
```

We recall that `\tl_if_in:nnTF` is slightly faster than `\str_if_in:nnTF`.

```
499 \tl_if_in:nnTF { ##1 } { - }
500 { \@@_cut_on_hyphen:w ##1 \q_stop }
501 }
```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
502 \def \l_tmpa_tl { ##1 }
503 \def \l_tmpb_tl { ##1 }
504 }
505 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
506 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
507 }
508 \tl_set_eq:NN #1 \l_tmpa_clist
509 }
510 }
```

The following internal parameters are for:

- `\Ldots` with both *extremities open* (and hence also `\Hdotsfor` in an exterior row;
- when the special character “:” is used in order to put the label of a so-called “dotted line” *on the line*, a margin of `\c_@@_innersep_middle_dim` will be added around the label.

```
511 \hook_gput_code:nnn { begindocument } { . }
512 {
513 \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
514 \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
515 }
```

5 The command `\tabularnote`

Of course, it's possible to use `\tabularnote` in the main tabular. But there is also the possibility to use that command in the caption of the tabular. And the caption may be specified by two means:

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It's also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that's mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That's why:
 - The number of tabular notes present in the caption will be written on the aux file and available in `\g_@@_notes_caption_int`.⁴
 - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker expressed by `\c_novalue_tl`).
 - During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
 - After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

The LaTeX counter `tabularnote` will be used to count the tabular notes during the construction of the array (this counter won't be used during the composition of the notes at the end of the array). You use a LaTeX counter because we will use `\refstepcounter` in order to have the tabular notes referenceable.

```
516 \newcounter { tabularnote }
```

We want to avoid error messages for duplicate labels when the package `hyperref` is used. That's why we will count all the tabular notes of the whole document with `\g_@@_tabularnote_int`.

```
517 \int_new:N \g_@@_tabularnote_int
```

```
518 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
```

```
519 \seq_new:N \g_@@_notes_seq
```

```
520 \seq_new:N \g_@@_notes_in_caption_seq
```

Before the actual tabular notes, it's possible to put a text specified by the key `tabularnote` of the environment. The token list `\g_@@_tabularnote_tl` corresponds to the value of that key.

```
521 \tl_new:N \g_@@_tabularnote_tl
```

We prepare the tools for the formatting of the references of the footnotes (in the tabular itself). There may have several references of footnote at the same point and we have to take into account that point.

```
522 \seq_new:N \l_@@_notes_labels_seq
```

```
523 \newcounter { nicematrix_draft }
```

⁴More precisely, it's the number of tabular notes which do not use the optional argument of `\tabularnote`.

```

524 \cs_new_protected:Npn \@@_notes_format:n #1
525 {
526   \setcounter { nicematrix_draft } { #1 }
527   \@@_notes_style:n { nicematrix_draft }
528 }

```

The following function can be redefined by using the key `notes/style`.

```

529 \cs_new:Npn \@@_notes_style:n #1 { \textit { \alph { #1 } } }

```

The following function can be redefined by using the key `notes/label-in-tabular`.

```

530 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }

```

The following function can be redefined by using the key `notes/label-in-list`.

```

531 \cs_new:Npn \@@_notes_label_in_list:n #1 { \textsuperscript { #1 } }

```

We define `\thetabularnote` because it will be used by LaTeX if the user want to reference a tabular which has been marked by a `\label`. The TeX group is for the case where the user has put an instruction such as `\color{red}` in `\@@_notes_style:n`.

```

532 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }

```

The tabular notes will be available for the final user only when `enumitem` is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by `enumitem` (a list `tabularnotes` in the general case and a list `tabularnotes*` if the key `para` is in force). However, we can test whether `enumitem` has been loaded only at the beginning of the document (we want to allow the user to load `enumitem` after `nicematrix`).

```

533 \hook_gput_code:nnn { begindocument } { . }
534 {
535   \IfPackageLoadedTF { enumitem }
536   {

```

The type of list `tabularnotes` will be used to format the tabular notes at the end of the array in the general case and `tabularnotes*` will be used if the key `para` is in force.

```

537     \newlist { tabularnotes } { enumerate } { 1 }
538     \setlist [ tabularnotes ]
539     {
540       topsep = \c_zero_dim ,
541       noitemsep ,
542       leftmargin = * ,
543       align = left ,
544       labelsep = \c_zero_dim ,
545       label =
546         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
547     }
548     \newlist { tabularnotes* } { enumerate* } { 1 }
549     \setlist* [ tabularnotes* ]
550     {
551       afterlabel = \nobreak ,
552       itemjoin = \quad ,
553       label =
554         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
555     }

```

One must remind that we have allowed a `\tabular` in the caption and that caption may also be found in the list of tables (`\listoftables`). We want the command `\tabularnote` be no-op during the composition of that list. That's why we program `\tabularnote` to be no-op excepted in a floating environment or in an environment of `nicematrix`.

```

556     \NewDocumentCommand \tabularnote { o m }
557     {
558       \bool_lazy_or:nnT { \cs_if_exist_p:N \capttype } { \l_@@_in_env_bool }

```

```

559     {
560     \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
561     { \@@_error:n { tabularnote~forbidden } }
562     {
563     \bool_if:NTF \l_@@_in_caption_bool
564     \@@_tabularnote_caption:nn
565     \@@_tabularnote:nn
566     { #1 } { #2 }
567     }
568     }
569     }
570   }
571   {
572   \NewDocumentCommand \tabularnote { o m }
573   { \@@_err_enumitem_not_loaded: }
574   }
575 }
576 \cs_new_protected:Npn \@@_err_enumitem_not_loaded:
577 {
578 \@@_error_or_warning:n { enumitem~not~loaded }
579 \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
580 }
581 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
582 { \tl_if_novalue:nT { #1 } { #3 } }

```

For the version in normal conditions, that is to say not in the caption. #1 is the optional argument of `\tabularnote` (maybe equal to the special marker expressed by `\c_novalue_tl`) and #2 is the mandatory argument of `\tabularnote`.

```

583 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
584 {

```

You have to see whether the argument of `\tabularnote` has yet been used as argument of another `\tabularnote` in the same tabular. In that case, there will be only one note (for both commands `\tabularnote`) at the end of the tabular. We search the argument of our command `\tabularnote` in `\g_@@_notes_seq`. The position in the sequence will be stored in `\l_tmpa_int` (0 if the text is not in the sequence yet).

```

585 \int_zero:N \l_tmpa_int
586 \bool_if:NT \l_@@_notes_detect_duplicates_bool
587 {

```

We recall that each component of `\g_@@_notes_seq` is a kind of couple of the form

`{label}{text of the tabularnote}`.

If the user have used `\tabularnote` without the optional argument, the *label* will be the special marker expressed by `\c_novalue_tl`.

When we will go through the sequence `\g_@@_notes_seq`, we will count in `\l_tmpb_int` the notes without explicit label in order to have the “current” value of the counter `\c@tabularnote`.

```

588 \int_zero:N \l_tmpb_int
589 \seq_map_indexed_inline:Nn \g_@@_notes_seq
590 {
591 \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
592 \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
593 {
594 \tl_if_novalue:nTF { #1 }
595 { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
596 { \int_set:Nn \l_tmpa_int { ##1 } }
597 \seq_map_break:
598 }
599 }
600 \int_if_zero:nF { \l_tmpa_int }
601 { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }

```

```

602     }
603 \int_if_zero:nT { \l_tmpa_int }
604 {
605     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
606     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
607 }
608 \seq_put_right:Ne \l_@@_notes_labels_seq
609 {
610     \tl_if_novalue:nTF { #1 }
611     {
612         \@@_notes_format:n
613         {
614             \int_eval:n
615             {
616                 \int_if_zero:nTF { \l_tmpa_int }
617                 { \c@tabularnote }
618                 { \l_tmpa_int }
619             }
620         }
621     }
622     { #1 }
623 }
624 \peek_meaning:NF \tabularnote
625 {

```

If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell when `\l_@@_hpos_cell_tl` is equal to `c` or `r`.

```

626     \hbox_set:Nn \l_tmpa_box
627     {

```

We remind that it is the command `\@@_notes_label_in_tabular:n` that will put the labels in a `\textsuperscript`.

```

628         \@@_notes_label_in_tabular:n
629         {
630             \seq_use:Nnnn
631             \l_@@_notes_labels_seq { , } { , } { , }
632         }
633     }

```

We want the (last) tabular note referenceable (with the standard command `\label`).

```

634     \int_gdecr:N \c@tabularnote
635     \int_set_eq:NN \l_tmpa_int \c@tabularnote

```

The following line is only to avoid error messages for multiply defined labels when the package `hyperref` is used.

```

636     \int_gincr:N \g_@@_tabularnote_int
637     \refstepcounter { tabularnote }
638     \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
639     { \int_gincr:N \c@tabularnote }
640     \seq_clear:N \l_@@_notes_labels_seq
641     \bool_lazy_or:nnTF
642     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
643     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
644     {
645         \hbox_overlap_right:n { \box_use:N \l_tmpa_box }

```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array`?) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```

646         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
647     }
648     { \box_use:N \l_tmpa_box }
649 }

```

```
650 }
```

Now the version when the command is used in the key `caption`. The main difficulty is that the argument of the command `\caption` is composed several times. In order to know the number of commands `\tabularnote` in the caption, we will consider that there should not be the same tabular note twice in the caption (in the main tabular, it's possible). Once we have found a tabular note which has yet been encountered, we consider that you are in a new composition of the argument of `\caption`.

```
651 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
652 {
653   \bool_if:NTF \g_@@_caption_finished_bool
654   {
655     \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
656     { \int_gzero:N \c@tabularnote }
```

Now, we try to detect duplicate notes in the caption. Be careful! We must put `\tl_if_in:NnF` and not `\tl_if_in:NnT`!

```
657   \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
658   { \@@_error:n { Identical~notes~in~caption } }
659 }
660 {
```

In the following code, we are in the first composition of the caption or at the first `\tabularnote` of the second composition.

```
661   \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
662   {
```

Now, we know that are in the second composition of the caption since we are reading a tabular note which has yet been read. Now, the value of `\g_@@_notes_caption_int` won't change anymore: it's the number of uses *without optional argument* of the command `\tabularnote` in the caption.

```
663     \bool_gset_true:N \g_@@_caption_finished_bool
664     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
665     \int_gzero:N \c@tabularnote
666   }
667   { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
668 }
```

Now, we will compose the label of the footnote (in the caption). Even if we are not in the first composition, we have to compose that label!

```
669   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
670   \seq_put_right:Ne \l_@@_notes_labels_seq
671   {
672     \tl_if_novalue:nTF { #1 }
673     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
674     { #1 }
675   }
676   \peek_meaning:NF \tabularnote
677   {
678     \@@_notes_label_in_tabular:n
679     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
680     \seq_clear:N \l_@@_notes_labels_seq
681   }
682 }

683 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
684 { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }
```

6 Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).

#1 is the name of the node which will be created; #2 and #3 are the coordinates of one of the corner of the rectangle; #4 and #5 are the coordinates of the opposite corner.

```

685 \cs_new_protected:Npn \@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
686 {
687   \begin { pgfscope }
688   \pgfset
689   {
690     inner~sep = \c_zero_dim ,
691     minimum~size = \c_zero_dim
692   }
693   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
694   \pgfnode
695   { rectangle }
696   { center }
697   {
698     \vbox_to_ht:nn
699     { \dim_abs:n { #5 - #3 } }
700     {
701       \vfill
702       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
703     }
704   }
705   { #1 }
706   { }
707   \end { pgfscope }
708 }

```

The command \@_pgf_rect_node:nnn is a variant of \@_pgf_rect_node:nnnnn: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

```

709 \cs_new_protected:Npn \@_pgf_rect_node:nnn #1 #2 #3
710 {
711   \begin { pgfscope }
712   \pgfset
713   {
714     inner~sep = \c_zero_dim ,
715     minimum~size = \c_zero_dim
716   }
717   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
718   \pgfpointdiff { #3 } { #2 }
719   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
720   \pgfnode
721   { rectangle }
722   { center }
723   {
724     \vbox_to_ht:nn
725     { \dim_abs:n \l_tmpb_dim }
726     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
727   }
728   { #1 }
729   { }
730   \end { pgfscope }
731 }

```

7 The options

The following parameter corresponds to the keys `caption`, `short-caption` and `label` of the environment `{NiceTabular}`.

```

732 \tl_new:N \l_@_caption_tl

```

```

733 \tl_new:N \l_@@_short_caption_tl
734 \tl_new:N \l_@@_label_tl

```

The following parameter corresponds to the key `caption-above` of `\NiceMatrixOptions`. When this parameter is `true`, the captions of the environments `{NiceTabular}`, specified with the key `caption` are put above the tabular (and below elsewhere).

```

735 \bool_new:N \l_@@_caption_above_bool

```

By default, the behaviour of `\cline` is changed in the environments of `nicematrix`: a `\cline` spreads the array by an amount equal to `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

```

736 \bool_new:N \l_@@_standard_cline_bool

```

The following dimensions correspond to the options `cell-space-top-limit` and `co` (these parameters are inspired by the package `cellspace`).

```

737 \dim_new:N \l_@@_cell_space_top_limit_dim
738 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

The following parameter corresponds to the key `xdots/horizontal_labels`.

```

739 \bool_new:N \l_@@_xdots_h_labels_bool

```

The following dimension is the distance between two dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.45 em but it will be changed if the option `small` is used.

```

740 \dim_new:N \l_@@_xdots_inter_dim
741 \hook_gput_code:nnn { begindocument } { . }
742 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }

```

The unit is `em` and that's why we fix the dimension after the preamble.

The following dimension is the distance between a node (in fact an anchor of that node) and a dotted line (for real dotted lines, the actual distance may, of course, be a bit larger, depending of the exact position of the dots).

```

743 \dim_new:N \l_@@_xdots_shorten_start_dim
744 \dim_new:N \l_@@_xdots_shorten_end_dim
745 \hook_gput_code:nnn { begindocument } { . }
746 {
747   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
748   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
749 }

```

The unit is `em` and that's why we fix the dimension after the preamble.

The following dimension is the radius of the dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.53 pt but it will be changed if the option `small` is used.

```

750 \dim_new:N \l_@@_xdots_radius_dim
751 \hook_gput_code:nnn { begindocument } { . }
752 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }

```

The unit is `em` and that's why we fix the dimension after the preamble.

The token list `\l_@@_xdots_line_style_tl` corresponds to the option `tikz` of the commands `\Cdots`, `\Ldots`, etc. and of the options `line-style` for the environments and `\NiceMatrixOptions`. The constant `\c_@@_standard_tl` will be used in some tests.

```

753 \tl_new:N \l_@@_xdots_line_style_tl
754 \tl_const:Nn \c_@@_standard_tl { standard }
755 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl

```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax` and the boolean `\l_@@_light_syntax_expanded_bool` correspond to the the option `light-syntax-expanded`.

```
756 \bool_new:N \l_@@_light_syntax_bool
757 \bool_new:N \l_@@_light_syntax_expanded_bool
```

The string `\l_@@_baseline_tl` may contain one of the three values `t`, `c` or `b` as in the option of the environment `{array}`. However, it may also contain **an integer** (which represents the number of the row to which align the array).

```
758 \tl_new:N \l_@@_baseline_tl
759 \tl_set:Nn \l_@@_baseline_tl { c }
```

The following parameter corresponds to the key `ampersand-in-blocks`

```
760 \bool_new:N \l_@@_amp_in_blocks_bool
```

The flag `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (as it is done in `{array}` of `array`).

```
761 \bool_new:N \l_@@_exterior_arraycolsep_bool
```

The flag `\l_@@_parallelize_diags_bool` controls whether the diagonals are parallelized. The initial value is `true`.

```
762 \bool_new:N \l_@@_parallelize_diags_bool
763 \bool_set_true:N \l_@@_parallelize_diags_bool
```

The following parameter correspond to the key `corners`. The elements of that `clist` must be within `NW`, `SW`, `NE` and `SE`.

```
764 \clist_new:N \l_@@_corners_clist
```

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

```
765 \bool_new:N \l_@@_nullify_dots_bool
```

When the key `respect-arraystretch` is used, the following command will be nullified.

```
766 \cs_new_protected:Npn \@@_reset_arraystretch: { \def \arraystretch { 1 } }
```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

```
767 \bool_new:N \l_@@_auto_columns_width_bool
```

The following boolean corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the “cell nodes” will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```
768 \bool_new:N \g_@@_create_cell_nodes_bool
```

The string `\l_@@_name_str` will contain the optional name of the environment: this name can be used to access to the Tikz nodes created in the array from outside the environment.

```
769 \str_new:N \l_@@_name_str
```

The boolean `\l_@@_medium_nodes_bool` will be used to indicate whether the “medium nodes” are created in the array. Idem for the “large nodes”.

```
770 \bool_new:N \l_@@_medium_nodes_bool
771 \bool_new:N \l_@@_large_nodes_bool
```

The boolean `\l_@@_except_borders_bool` will be raised when the key `hvlines-except-borders` will be used (but that key has also other effects).

```
772 \bool_new:N \l_@@_except_borders_bool
```

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

```
773 \dim_new:N \l_@@_left_margin_dim
774 \dim_new:N \l_@@_right_margin_dim
```

The dimensions `\l_@@_extra_left_margin_dim` and `\l_@@_extra_right_margin_dim` correspond to the options `extra-left-margin` and `extra-right-margin`.

```
775 \dim_new:N \l_@@_extra_left_margin_dim
776 \dim_new:N \l_@@_extra_right_margin_dim
```

The token list `\l_@@_end_of_row_tl` corresponds to the option `end-of-row`. It specifies the symbol used to mark the ends of rows when the light syntax is used.

```
777 \tl_new:N \l_@@_end_of_row_tl
778 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

The following parameter is for the color the dotted lines drawn by `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` but *not* the dotted lines drawn by `\hdottedline` and “:”.

```
779 \tl_new:N \l_@@_xdots_color_tl
```

The following token list corresponds to the key `delimiters/color`.

```
780 \tl_new:N \l_@@_delimiters_color_tl
```

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option `columns-width` or the option `auto-columns-width` of the environment `{NiceMatrixBlock}`). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is fonction of its size. That’s why we create an option called `delimiters/max-width` which will give to the delimiters the width of a delimiter (of the same type) of big size. The following boolean corresponds to this option.

```
781 \bool_new:N \l_@@_delimiters_max_width_bool
```

```
782 \keys_define:nn { nicematrix / xdots }
783 {
784   Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
785   shorten-start .code:n =
786     \hook_gput_code:nnn { begindocument } { . }
787     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
788   shorten-end .code:n =
789     \hook_gput_code:nnn { begindocument } { . }
790     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
791   shorten-start .value_required:n = true ,
792   shorten-end .value_required:n = true ,
793   shorten .code:n =
794     \hook_gput_code:nnn { begindocument } { . }
795     {
796       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
797       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
798     } ,
799   shorten .value_required:n = true ,
800   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
801   horizontal-labels .default:n = true ,
802   horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
803   horizontal-label .default:n = true ,
804   line-style .code:n =
```

```

805     {
806       \bool_lazy_or:nnTF
807         { \cs_if_exist_p:N \tikzpicture }
808         { \str_if_eq_p:nn { #1 } { standard } }
809         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
810         { \@@_error:n { bad-option-for-line-style } }
811     } ,
812     line-style .value_required:n = true ,
813     color .tl_set:N = \l_@@_xdots_color_tl ,
814     color .value_required:n = true ,
815     radius .code:n =
816       \hook_gput_code:nnn { begindocument } { . }
817       { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
818     radius .value_required:n = true ,
819     inter .code:n =
820       \hook_gput_code:nnn { begindocument } { . }
821       { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
822     radius .value_required:n = true ,

```

The options `down`, `up` and `middle` are not documented for the final user because he should use the syntax with `^`, `_` and `:`. We use `\tl_put_right:Nn` and not `\tl_set:Nn` (or `.tl_set:N`) because we don't want a direct use of `up=...` erased by an absent `^{\dots}`.

```

823     down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
824     up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
825     middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

The key `draw-first`, which is meant to be used only with `\Ddots` and `\Iddots`, will be caught when `\Ddots` or `\Iddots` is used (during the construction of the array and not when we draw the dotted lines).

```

826     draw-first .code:n = \prg_do_nothing: ,
827     unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
828   }

```

```

829 \keys_define:nn { nicematrix / rules }
830   {
831     color .tl_set:N = \l_@@_rules_color_tl ,
832     color .value_required:n = true ,
833     width .dim_set:N = \arrayrulewidth ,
834     width .value_required:n = true ,
835     unknown .code:n = \@@_error:n { Unknown~key~for~rules }
836   }
837 \cs_new_protected:Npn \@@_err_key_color_inside:
838   {
839     \@@_error_or_warning:n { key-color-inside }
840     \cs_gset:Npn \@@_err_key_color_inside: { }
841   }

```

First, we define a set of keys “`nicematrix / Global`” which will be used (with the mechanism of `.inherit:n`) by other sets of keys.

```

842 \keys_define:nn { nicematrix / Global }
843   {
844     color-inside .code:n = \@@_err_key_color_inside: ,
845     colortbl-like .code:n = \@@_err_key_color_inside: ,
846     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
847     ampersand-in-blocks .default:n = true ,
848     &-in-blocks .meta:n = ampersand-in-blocks ,
849     no-cell-nodes .code:n =
850       \bool_set_true:N \l_@@_no_cell_nodes_bool
851       \cs_set_protected:Npn \@@_node_cell:
852         { \set@color \box_use_drop:N \l_@@_cell_box } ,
853     no-cell-nodes .value_forbidden:n = true ,
854     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,

```

```

855 rounded-corners .default:n = 4 pt ,
856 custom-line .code:n = \@@_custom_line:n { #1 } ,
857 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
858 rules .value_required:n = true ,
859 standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
860 standard-cline .default:n = true ,
861 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
862 cell-space-top-limit .value_required:n = true ,
863 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
864 cell-space-bottom-limit .value_required:n = true ,
865 cell-space-limits .meta:n =
866 {
867   cell-space-top-limit = #1 ,
868   cell-space-bottom-limit = #1 ,
869 } ,
870 cell-space-limits .value_required:n = true ,
871 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
872 light-syntax .code:n =
873   \bool_set_true:N \l_@@_light_syntax_bool
874   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
875 light-syntax .value_forbidden:n = true ,
876 light-syntax-expanded .code:n =
877   \bool_set_true:N \l_@@_light_syntax_bool
878   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
879 light-syntax-expanded .value_forbidden:n = true ,
880 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
881 end-of-row .value_required:n = true ,
882 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
883 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
884 last-row .int_set:N = \l_@@_last_row_int ,
885 last-row .default:n = -1 ,
886 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
887 code-for-first-col .value_required:n = true ,
888 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
889 code-for-last-col .value_required:n = true ,
890 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
891 code-for-first-row .value_required:n = true ,
892 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
893 code-for-last-row .value_required:n = true ,
894 hlines .clist_set:N = \l_@@_hlines_clist ,
895 vlines .clist_set:N = \l_@@_vlines_clist ,
896 hlines .default:n = all ,
897 vlines .default:n = all ,
898 vlines-in-sub-matrix .code:n =
899 {
900   \tl_if_single_token:nTF { #1 }
901   {
902     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
903     { \@@_error:nn { Forbidden~letter } { #1 } }

```

We write directly a command for the automata which reads the preamble provided by the final user.

```

904   { \cs_set_eq:cN { @@_#1 : } \@@_make_preamble_vlism:n }
905   }
906   { \@@_error:n { One~letter~allowed } }
907   } ,
908 vlines-in-sub-matrix .value_required:n = true ,
909 hvlines .code:n =
910 {
911   \bool_set_true:N \l_@@_hvlines_bool
912   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
913   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
914 } ,
915 hvlines-except-borders .code:n =
916 {

```

```

917     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
918     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
919     \bool_set_true:N \l_@@_hvlines_bool
920     \bool_set_true:N \l_@@_except_borders_bool
921   } ,
922   parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

923   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
924   renew-dots .value_forbidden:n = true ,
925   nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
926   create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
927   create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
928   create-extra-nodes .meta:n =
929     { create-medium-nodes , create-large-nodes } ,
930   left-margin .dim_set:N = \l_@@_left_margin_dim ,
931   left-margin .default:n = \arraycolsep ,
932   right-margin .dim_set:N = \l_@@_right_margin_dim ,
933   right-margin .default:n = \arraycolsep ,
934   margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
935   margin .default:n = \arraycolsep ,
936   extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
937   extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
938   extra-margin .meta:n =
939     { extra-left-margin = #1 , extra-right-margin = #1 } ,
940   extra-margin .value_required:n = true ,
941   respect-arraystretch .code:n =
942     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
943   respect-arraystretch .value_forbidden:n = true ,
944   pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
945   pgf-node-code .value_required:n = true
946 }

```

We define a set of keys used by the environments of `nicematrix` (but not by the command `\NiceMatrixOptions`).

```

947 \keys_define:nn { nicematrix / environments }
948 {
949   corners .clist_set:N = \l_@@_corners_clist ,
950   corners .default:n = { NW , SW , NE , SE } ,
951   code-before .code:n =
952     {
953       \tl_if_empty:nF { #1 }
954       {
955         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
956         \bool_set_true:N \l_@@_code_before_bool
957       }
958     } ,
959   code-before .value_required:n = true ,

```

The options `c`, `t` and `b` of the environment `{NiceArray}` have the same meaning as the option of the classical environment `{array}`.

```

960   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
961   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
962   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
963   baseline .tl_set:N = \l_@@_baseline_tl ,
964   baseline .value_required:n = true ,
965   columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

966   \str_if_eq:eeTF { #1 } { auto }

```

```

967     { \bool_set_true:N \l_@@_auto_columns_width_bool }
968     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
969     columns-width .value_required:n = true ,
970     name .code:n =

```

We test whether we are in the measuring phase of an environment of `amsmath` (always loaded by `nicematrix`) because we want to avoid a fallacious message of duplicate name in this case.

```

971     \legacy_if:nF { measuring@ }
972     {
973         \str_set:Ne \l_@@_name_str { #1 }
974         \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
975         { \@@_err_duplicate_names:n { #1 } }
976         { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
977     } ,
978     name .value_required:n = true ,
979     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
980     code-after .value_required:n = true ,
981 }

982 \cs_set:Npn \@@_err_duplicate_names:n #1
983 { \@@_error:nn { Duplicate-name } { #1 } }

984 \keys_define:nn { nicematrix / notes }
985 {
986     para .bool_set:N = \l_@@_notes_para_bool ,
987     para .default:n = true ,
988     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
989     code-before .value_required:n = true ,
990     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
991     code-after .value_required:n = true ,
992     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
993     bottomrule .default:n = true ,
994     style .cs_set:Np = \@@_notes_style:n #1 ,
995     style .value_required:n = true ,
996     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
997     label-in-tabular .value_required:n = true ,
998     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
999     label-in-list .value_required:n = true ,
1000     enumitem-keys .code:n =
1001     {
1002         \hook_gput_code:nnn { begindocument } { . }
1003         {
1004             \IfPackageLoadedT { enumitem }
1005             { \setlist* [ tabularnotes ] { #1 } }
1006         }
1007     } ,
1008     enumitem-keys .value_required:n = true ,
1009     enumitem-keys-para .code:n =
1010     {
1011         \hook_gput_code:nnn { begindocument } { . }
1012         {
1013             \IfPackageLoadedT { enumitem }
1014             { \setlist* [ tabularnotes* ] { #1 } }
1015         }
1016     } ,
1017     enumitem-keys-para .value_required:n = true ,
1018     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1019     detect-duplicates .default:n = true ,
1020     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1021 }

1022 \keys_define:nn { nicematrix / delimiters }
1023 {
1024     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1025     max-width .default:n = true ,
1026     color .tl_set:N = \l_@@_delimiters_color_tl ,

```



```

1027     color .value_required:n = true ,
1028 }

```

We begin the construction of the major sets of keys (used by the different user commands and environments).

```

1029 \keys_define:nn { nicematrix }
1030 {
1031   NiceMatrixOptions .inherit:n =
1032     { nicematrix / Global } ,
1033   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1034   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1035   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1036   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1037   SubMatrix / rules .inherit:n = nicematrix / rules ,
1038   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1039   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1040   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1041   NiceMatrix .inherit:n =
1042     {
1043       nicematrix / Global ,
1044       nicematrix / environments ,
1045     } ,
1046   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1047   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1048   NiceTabular .inherit:n =
1049     {
1050       nicematrix / Global ,
1051       nicematrix / environments
1052     } ,
1053   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1054   NiceTabular / rules .inherit:n = nicematrix / rules ,
1055   NiceTabular / notes .inherit:n = nicematrix / notes ,
1056   NiceArray .inherit:n =
1057     {
1058       nicematrix / Global ,
1059       nicematrix / environments ,
1060     } ,
1061   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1062   NiceArray / rules .inherit:n = nicematrix / rules ,
1063   pNiceArray .inherit:n =
1064     {
1065       nicematrix / Global ,
1066       nicematrix / environments ,
1067     } ,
1068   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1069   pNiceArray / rules .inherit:n = nicematrix / rules ,
1070 }

```

We finalise the definition of the set of keys “nicematrix / NiceMatrixOptions” with the options specific to \NiceMatrixOptions.

```

1071 \keys_define:nn { nicematrix / NiceMatrixOptions }
1072 {
1073   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1074   delimiters / color .value_required:n = true ,
1075   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1076   delimiters / max-width .default:n = true ,
1077   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1078   delimiters .value_required:n = true ,
1079   width .dim_set:N = \l_@@_width_dim ,
1080   width .value_required:n = true ,
1081   last-col .code:n =
1082     \tl_if_empty:nF { #1 }

```

```

1083     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1084     \int_zero:N \l_@@_last_col_int ,
1085     small .bool_set:N = \l_@@_small_bool ,
1086     small .value_forbidden:n = true ,

```

With the option `renew-matrix`, the environment `{matrix}` of `amsmath` and its variants are redefined to behave like the environment `{NiceMatrix}` and its variants.

```

1087     renew-matrix .code:n = \@@_renew_matrix: ,
1088     renew-matrix .value_forbidden:n = true ,

```

The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```

1089     exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,

```

If the option `columns-width` is used, all the columns will have the same width.

In `\NiceMatrixOptions`, the special value `auto` is not available.

```

1090     columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

1091     \str_if_eq:eeTF { #1 } { auto }
1092     { \@@_error:n { Option-auto-for-columns-width } }
1093     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

Usually, an error is raised when the user tries to give the same name to two distinct environments of `nicematrix` (these names are global and not local to the current TeX scope). However, the option `allow-duplicate-names` disables this feature.

```

1094     allow-duplicate-names .code:n =
1095     \cs_set:Nn \@@_err_duplicate_names:n { } ,
1096     allow-duplicate-names .value_forbidden:n = true ,
1097     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1098     notes .value_required:n = true ,
1099     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1100     sub-matrix .value_required:n = true ,
1101     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1102     matrix / columns-type .value_required:n = true ,
1103     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1104     caption-above .default:n = true ,
1105     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1106 }

```

`\NiceMatrixOptions` is the command of the `nicematrix` package to fix options at the document level. The scope of these specifications is the current TeX group.

```

1107 \NewDocumentCommand \NiceMatrixOptions { m }
1108 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }

```

We finalise the definition of the set of keys “`nicematrix / NiceMatrix`”. That set of keys will be used by `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

```

1109 \keys_define:nn { nicematrix / NiceMatrix }
1110 {
1111     last-col .code:n = \tl_if_empty:nTF { #1 }
1112     {
1113         \bool_set_true:N \l_@@_last_col_without_value_bool
1114         \int_set:Nn \l_@@_last_col_int { -1 }
1115     }
1116     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1117     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1118     columns-type .value_required:n = true ,
1119     l .meta:n = { columns-type = l } ,
1120     r .meta:n = { columns-type = r } ,

```

```

1121 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1122 delimiters / color .value_required:n = true ,
1123 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1124 delimiters / max-width .default:n = true ,
1125 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1126 delimiters .value_required:n = true ,
1127 small .bool_set:N = \l_@@_small_bool ,
1128 small .value_forbidden:n = true ,
1129 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1130 }

```

We finalise the definition of the set of keys “nicematrix / NiceArray” with the options specific to {NiceArray}.

```

1131 \keys_define:nn { nicematrix / NiceArray }
1132 {

```

In the environments {NiceArray} and its variants, the option last-col must be used without value because the number of columns of the array is read from the preamble of the array.

```

1133 small .bool_set:N = \l_@@_small_bool ,
1134 small .value_forbidden:n = true ,
1135 last-col .code:n = \tl_if_empty:nF { #1 }
1136 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1137 \int_zero:N \l_@@_last_col_int ,
1138 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1139 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1140 unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1141 }

```

```

1142 \keys_define:nn { nicematrix / pNiceArray }
1143 {
1144 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1145 last-col .code:n = \tl_if_empty:nF { #1 }
1146 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1147 \int_zero:N \l_@@_last_col_int ,
1148 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1149 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1150 delimiters / color .value_required:n = true ,
1151 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1152 delimiters / max-width .default:n = true ,
1153 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1154 delimiters .value_required:n = true ,
1155 small .bool_set:N = \l_@@_small_bool ,
1156 small .value_forbidden:n = true ,
1157 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1158 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1159 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1160 }

```

We finalise the definition of the set of keys “nicematrix / NiceTabular” with the options specific to {NiceTabular}.

```

1161 \keys_define:nn { nicematrix / NiceTabular }
1162 {

```

The dimension width will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

```

1163 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1164 \bool_set_true:N \l_@@_width_used_bool ,
1165 width .value_required:n = true ,
1166 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1167 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1168 tabularnote .value_required:n = true ,
1169 caption .tl_set:N = \l_@@_caption_tl ,

```

```

1170   caption .value_required:n = true ,
1171   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1172   short-caption .value_required:n = true ,
1173   label .tl_set:N = \l_@@_label_tl ,
1174   label .value_required:n = true ,
1175   last-col .code:n = \tl_if_empty:nF { #1 }
1176     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1177     \int_zero:N \l_@@_last_col_int ,
1178   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1179   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1180   unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1181 }

```

The `\CodeAfter` (inserted with the key `code-after` or after the keyword `\CodeAfter`) may always begin with a list of pairs `key=value` between square brackets. Here is the corresponding set of keys. We *must* put the following instructions *after* the :

```
CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix
```

```

1182 \keys_define:nn { nicematrix / CodeAfter }
1183 {
1184   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1185   delimiters / color .value_required:n = true ,
1186   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1187   rules .value_required:n = true ,
1188   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1189   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1190   sub-matrix .value_required:n = true ,
1191   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1192 }

```

8 Important code used by `{NiceArrayWithDelims}`

The pseudo-environment `\@@_cell_begin:-\@@_cell_end:` will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a `\halign` (via an environment `{array}`).

```

1193 \cs_new_protected:Npn \@@_cell_begin:
1194 {

```

`\g_@@_cell_after_hook_tl` will be set during the composition of the box `\l_@@_cell_box` and will be used *after* the composition in order to modify that box.

```
1195   \tl_gclear:N \g_@@_cell_after_hook_tl
```

At the beginning of the cell, we link `\CodeAfter` to a command which do begin with `\` (whereas the standard version of `\CodeAfter` does not).

```
1196   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

The following link only to have a better error message when `\Hline` is used in another place than the beginning of a line.

```
1197   \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

We increment the LaTeX counter `jCol`, which is the counter of the columns.

```
1198   \int_gincr:N \c@jCol
```

Now, we increment the counter of the rows. We don't do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don't want to take into account.

```

1199   \int_compare:nNnT { \c@jCol } = { \c_one_int }
1200   {
1201     \int_compare:nNnT { \l_@@_first_col_int } = { \c_one_int }
1202     { \@@_begin_of_row: }
1203   }

```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` is in the `\@@_cell_end:`.

```

1204   \hbox_set:Nw \l_@@_cell_box

```

The following command is nullified in the tabulars.

```

1205   \@@_tuning_not_tabular_begin:
1206   \@@_tuning_first_row:
1207   \@@_tuning_last_row:
1208   \g_@@_row_style_tl
1209   }

```

The following command will be nullified unless there is a first row.

Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT { \c@iRow }
  {
    \int_if_zero:nF { \c@jCol }
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}

```

We will use a version a little more efficient.

```

1210 \cs_new_protected:Npn \@@_tuning_first_row:
1211   {
1212     \if_int_compare:w \c@iRow = \c_zero_int
1213     \if_int_compare:w \c@jCol > \c_zero_int
1214     \l_@@_code_for_first_row_tl
1215     \xglobal \colorlet { nicematrix-first-row } { . }
1216     \fi:
1217     \fi:
1218   }

```

The following command will be nullified unless there is a last row and we know its value (*ie*: `\l_@@_last_row_int > 0`).

```

\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}

```

We will use a version a little more efficient.

```

1219 \cs_new_protected:Npn \@@_tuning_last_row:
1220   {
1221     \if_int_compare:w \c@iRow = \l_@@_last_row_int
1222     \l_@@_code_for_last_row_tl

```

```

1223     \xglobal \colorlet { nicematrix-last-row } { . }
1224     \fi:
1225 }

```

A different value will be provided to the following commands when the key `small` is in force.

```

1226 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1227 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1228 {
1229     \m@th
1230     \c_math_toggle_token

```

A special value is provided by the following control sequence when the key `small` is in force.

```

1231     \@@_tuning_key_small:
1232 }
1233 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

The following macro `\@@_begin_of_row` is usually used in the cell number 1 of the row. However, when the key `first-col` is used, `\@@_begin_of_row` is executed in the cell number 0 of the row.

```

1234 \cs_new_protected:Npn \@@_begin_of_row:
1235 {
1236     \int_gincr:N \c@iRow
1237     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1238     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1239     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1240     \pgfpicture
1241     \pgfrememberpicturepositiononpagetrue
1242     \pgfcoordinate
1243     { \@@_env: - row - \int_use:N \c@iRow - base }
1244     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1245     \str_if_empty:NF \l_@@_name_str
1246     {
1247         \pgfnodealias
1248         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1249         { \@@_env: - row - \int_use:N \c@iRow - base }
1250     }
1251     \endpgfpicture
1252 }

```

Remark: If the key `create-cell-nodes` of the `\CodeBefore` is used, then we will add some lines to that command.

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give information about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command.

```

1253 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1254 {
1255     \int_if_zero:nTF { \c@iRow }
1256     {
1257         \dim_compare:nNnT
1258         { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
1259         { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1260         \dim_compare:nNnT
1261         { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
1262         { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1263     }
1264     {
1265         \int_compare:nNnT { \c@iRow } = { \c_one_int }
1266         {
1267             \dim_compare:nNnT
1268             { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
1269             { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }

```

```

1270     }
1271   }
1272 }
1273 \cs_new_protected:Npn \@@_rotate_cell_box:
1274 {
1275   \box_rotate:Nn \l_@@_cell_box { 90 }
1276   \bool_if:NTF \g_@@_rotate_c_bool
1277   {
1278     \hbox_set:Nn \l_@@_cell_box
1279     {
1280       \m@th
1281       \c_math_toggle_token
1282       \vcenter { \box_use:N \l_@@_cell_box }
1283       \c_math_toggle_token
1284     }
1285   }
1286   {
1287     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
1288     {
1289       \vbox_set_top:Nn \l_@@_cell_box
1290       {
1291         \vbox_to_zero:n { }
1292         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1293         \box_use:N \l_@@_cell_box
1294       }
1295     }
1296   }
1297   \bool_gset_false:N \g_@@_rotate_bool
1298   \bool_gset_false:N \g_@@_rotate_c_bool
1299 }
1300 \cs_new_protected:Npn \@@_adjust_size_box:
1301 {
1302   \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
1303   {
1304     \box_set_wd:Nn \l_@@_cell_box
1305     { \dim_max:nn { \box_wd:N \l_@@_cell_box } { \g_@@_blocks_wd_dim } }
1306     \dim_gzero:N \g_@@_blocks_wd_dim
1307   }
1308   \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
1309   {
1310     \box_set_dp:Nn \l_@@_cell_box
1311     { \dim_max:nn { \box_dp:N \l_@@_cell_box } { \g_@@_blocks_dp_dim } }
1312     \dim_gzero:N \g_@@_blocks_dp_dim
1313   }
1314   \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
1315   {
1316     \box_set_ht:Nn \l_@@_cell_box
1317     { \dim_max:nn { \box_ht:N \l_@@_cell_box } { \g_@@_blocks_ht_dim } }
1318     \dim_gzero:N \g_@@_blocks_ht_dim
1319   }
1320 }
1321 \cs_new_protected:Npn \@@_cell_end:
1322 {
The following command is nullified in the tabulars.
1323   \@@_tuning_not_tabular_end:
1324   \hbox_set_end:
1325   \@@_cell_end_i:
1326 }
1327 \cs_new_protected:Npn \@@_cell_end_i:
1328 {

```

The token list `\g_@@_cell_after_hook_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```

1329   \g_@@_cell_after_hook_tl
1330   \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
1331   \@@_adjust_size_box:

1332   \box_set_ht:Nn \l_@@_cell_box
1333     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1334   \box_set_dp:Nn \l_@@_cell_box
1335     { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

```

1336   \@@_update_max_cell_width:

```

The following computations are for the “first row” and the “last row”.

```

1337   \@@_update_for_first_and_last_row:

```

If the cell is empty, or may be considered as if, we must not create the PGF node, for two reasons:

- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

However, it’s difficult to determine whether a cell is empty. Up to now we use the following technique:

- for the columns of type p, m, b, V (of `\varwidth`) or X, we test whether the cell is syntactically empty with `\@@_test_if_empty:` and `\@@_test_if_empty_for_S:`
- if the width of the box `\l_@@_cell_box` (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a `\rlap`, `\llap`, `\clap` or a `\mathclap` of `mathtools`).
- the cells with a command `\Ldots` or `\Cdots`, `\Vdots`, etc., should also be considered as empty; if `\nullify-dots` is in force, there would be nothing to do (in this case the previous commands only write an instruction in a kind of `\CodeAfter`); however, if `\nullify-dots` is not in force, a phantom of `\ldots`, `\cdots`, `\vdots` is inserted and its width is not equal to zero; that’s why these commands raise a boolean `\g_@@_empty_cell_bool` and we begin by testing this boolean.

```

1338   \bool_if:NTF \g_@@_empty_cell_bool
1339     { \box_use_drop:N \l_@@_cell_box }
1340     {
1341       \bool_if:NTF \g_@@_not_empty_cell_bool
1342         { \@@_print_node_cell: }
1343         {
1344           \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
1345             { \@@_print_node_cell: }
1346             { \box_use_drop:N \l_@@_cell_box }
1347         }
1348     }
1349   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
1350     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1351   \bool_gset_false:N \g_@@_empty_cell_bool
1352   \bool_gset_false:N \g_@@_not_empty_cell_bool
1353 }

```

The following command will be nullified in our redefinition of `\multicolumn`.

```

1354 \cs_new_protected:Npn \@@_update_max_cell_width:
1355   {
1356     \dim_gset:Nn \g_@@_max_cell_width_dim
1357     { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
1358   }

```


The following variant of `\@@_cell_end:` is only for the columns of type `w{s}{...}` or `W{s}{...}` (which use the horizontal alignment key `s` of `\makebox`).

```

1359 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1360 {
1361   \@@_math_toggle:
1362   \hbox_set_end:
1363   \bool_if:NF \g_@@_rotate_bool
1364   {
1365     \hbox_set:Nn \l_@@_cell_box
1366     {
1367       \makebox [ \l_@@_col_width_dim ] [ s ]
1368       { \hbox_unpack_drop:N \l_@@_cell_box }
1369     }
1370   }
1371   \@@_cell_end_i:
1372 }

1373 \pgfset
1374 {
1375   nicematrix / cell-node /.style =
1376   {
1377     inner-sep = \c_zero_dim ,
1378     minimum-width = \c_zero_dim
1379   }
1380 }

```

In the cells of a column of type `S` (of `siunitx`), we have to wrap the command `\@@_node_cell:` inside a command of `siunitx` to enforce the correct horizontal alignment. In the cells of the columns with other columns type, we don't have to do that job. That's why we create a socket with its default plug (`identity`) and a plug when we have to do the wrapping.

```

1381 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1382 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1383 {
1384   \use:c
1385   {
1386     __siunitx_table_align_
1387     \bool_if:NTF \l_siunitx_table_text_bool
1388     { \l_siunitx_table_align_text_tl }
1389     { \l_siunitx_table_align_number_tl }
1390   }
1391 }
1392 { #1 }
1393 }

```

Now, a socket which deal with `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the “cell nodes” will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```

1394 \socket_new:nn { nicematrix / create-cell-nodes } { 1 }
1395 \socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
1396 {
1397   \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1398   \hbox:n
1399   {
1400     \pgfsys@markposition
1401     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
1402   }
1403   #1
1404   \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1405   \hbox:n
1406   {
1407     \pgfsys@markposition

```

```

1408         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1409     }
1410 }

1411 \cs_new_protected:Npn \@@_print_node_cell:
1412 {
1413     \socket_use:nn { nicematrix / siunitx-wrap }
1414     { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
1415 }

```

The following command creates the PGF name of the node with, of course, `\l_@@_cell_box` as the content.

```

1416 \cs_new_protected:Npn \@@_node_cell:
1417 {
1418     \pgfpicture
1419     \pgfsetbaseline \c_zero_dim
1420     \pgfrememberpicturepositiononpagetrue
1421     \pgfset { nicematrix / cell-node }
1422     \pgfnode
1423     { rectangle }
1424     { base }
1425     {

```

The following instruction `\set@color` has been added on 2022/10/06. It's necessary only with Xe-LaTeX and not with the other engines (we don't know why).

```

1426         \sys_if_engine_xetex:T { \set@color }
1427         \box_use:N \l_@@_cell_box
1428     }
1429     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1430     { \l_@@_pgf_node_code_tl }
1431     \str_if_empty:NF \l_@@_name_str
1432     {
1433         \pgfnodealias
1434         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1435         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1436     }
1437     \endpgfpicture
1438 }

```

The second argument of the following command `\@@_instruction_of_type:nnn` defined below is the type of the instruction (`Cdots`, `Vdots`, `Ddots`, etc.). The third argument is the list of options. This command writes in the corresponding `\g_@@_type_lines_tl` the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots[color=red] & & 
\end{pNiceMatrix}

```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & \cdots & & 6 \\ 7 & \cdots & & \end{pmatrix}$$

the content of `\g_@@_Cdots_lines_tl` will be:

```

\@@_draw_Cdots:nnn {2}{2}{}
\@@_draw_Cdots:nnn {3}{2}{color=red}

```

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions (with consequences for the parallelisation of the diagonal lines).

```

1439 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1440 {
1441     \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }

```

```

1442     { g_@@_ #2 _ lines _ t1 }
1443     {
1444       \use:c { @@ _ draw _ #2 : nmn }
1445       { \int_use:N \c@iRow }
1446       { \int_use:N \c@jCol }
1447       { \exp_not:n { #3 } }
1448     }
1449   }

1450 \cs_new_protected:Npn \@@_array:n
1451 {
1452   \dim_set:Nn \col@sep
1453   { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1454   \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
1455   { \def \@halignto { } }
1456   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

It colortbl is loaded, \@tabarray has been redefined to incorporate \CT@start.

```

1457   \@tabarray

```

\l_@@_baseline_t1 may have the value t, c or b. However, if the value is b, we compose the \array (of array) with the option t and the right translation will be done further. Remark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here. \str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).

```

1458   [ \str_if_eq:eeTF \l_@@_baseline_t1 { c } { c } { t } ]
1459   }
1460 \cs_generate_variant:Nn \@@_array:n { o }

```

We keep in memory the standard version of \ialign because we will redefine \ialign in the environment {NiceArrayWithDelims} but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), array uses \ar@ialign instead of \ialign. In that case, of course, you do a saving of \ar@ialign.

```

1461 \bool_if:nTF
1462 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }

```

We use here a \cs_set_eq:cN instead of a \cs_set_eq:NN in order to avoid a message when explcheck is used on nicematrix.sty.

```

1463 { \cs_set_eq:cN { @@_old_ar@ialign: } \ar@ialign }
1464 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

The following command creates a row node (and not a row of nodes!).

```

1465 \cs_new_protected:Npn \@@_create_row_node:
1466 {
1467   \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
1468   {
1469     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1470     \@@_create_row_node_i:
1471   }
1472 }

1473 \cs_new_protected:Npn \@@_create_row_node_i:
1474 {

```

The \hbox:n (or \hbox) is mandatory.

```

1475   \hbox
1476   {
1477     \bool_if:NT \l_@@_code_before_bool
1478     {
1479       \vtop
1480       {
1481         \skip_vertical:N 0.5\arrayrulewidth
1482         \pgfsys@markposition
1483         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } } }
1484         \skip_vertical:N -0.5\arrayrulewidth

```

```

1485     }
1486   }
1487   \pgfpicture
1488   \pgfrememberpicturepositiononpagetrue
1489   \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1490   { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1491   \str_if_empty:NF \l_@@_name_str
1492   {
1493     \pgfnodealias
1494     { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1495     { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1496   }
1497   \endpgfpicture
1498 }
1499 }

1500 \cs_new_protected:Npn \@@_in_everycr:
1501 {
1502   \bool_if:NT \c_@@_recent_array_bool
1503   {
1504     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1505     \tbl_update_cell_data_for_next_row:
1506   }
1507   \int_gzero:N \c@jCol
1508   \bool_gset_false:N \g_@@_after_col_zero_bool
1509   \bool_if:NF \g_@@_row_of_col_done_bool
1510   {
1511     \@@_create_row_node:

```

We don't draw now the rules of the key `hlines` (or `hvlines`) but we reserve the vertical space for these rules (the rules will be drawn by PGF).

```

1512   \clist_if_empty:NF \l_@@_hlines_clist
1513   {
1514     \str_if_eq:eeF \l_@@_hlines_clist { all }
1515     {
1516       \clist_if_in:NeT
1517       \l_@@_hlines_clist
1518       { \int_eval:n { \c@iRow + 1 } }
1519     }
1520   }

```

The counter `\c@iRow` has the value `-1` only if there is a “first row” and that we are before that “first row”, i.e. just before the beginning of the array.

```

1521     \int_compare:nNnT { \c@iRow } > { -1 }
1522     {
1523       \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
1524       { \hrule height \arrayrulewidth width \c_zero_dim }
1525     }
1526   }
1527 }
1528 }
1529 }

```

When the key `renew-dots` is used, the following code will be executed.

```

1530 \cs_set_protected:Npn \@@_renew_dots:
1531 {
1532   \cs_set_eq:NN \ldots \@@_Ldots:
1533   \cs_set_eq:NN \cdots \@@_Cdots:
1534   \cs_set_eq:NN \vdots \@@_Vdots:
1535   \cs_set_eq:NN \ddots \@@_Ddots:
1536   \cs_set_eq:NN \iddots \@@_Iddots:
1537   \cs_set_eq:NN \dots \@@_Ldots:

```

```

1538     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1539 }

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` before the vertical skip (and thus, at a wrong place). That's why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁵.

```

1540 \hook_gput_code:nnn { begindocument } { . }
1541 {
1542   \IfPackageLoadedTF { booktabs }
1543   {
1544     \cs_new_protected:Npn \@@_patch_booktabs:
1545       { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1546   }
1547   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1548 }

```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`⁶ and `\extrarowheight` (of `array`). That box is inserted (via `\@arstrut`) in the beginning of each row of the array. That's why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `\@arstrutbox` and that's why we do it in the `\ialign`.

```

1549 \cs_new_protected:Npn \@@_some_initialization:
1550 {
1551   \@@_everycr:
1552   \dim_gset:N \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1553   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1554   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1555   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1556   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1557   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1558 }

```

```

1559 \cs_new_protected:Npn \@@_pre_array_ii:
1560 {

```

The total weight of the letters `X` in the preamble of the array.

```

1561   \fp_gzero:N \g_@@_total_X_weight_fp
1562   \bool_gset_false:N \g_@@_V_of_X_bool
1563   \@@_expand_clist:N \l_@@_hlines_clist
1564   \@@_expand_clist:N \l_@@_vlines_clist
1565   \@@_patch_booktabs:
1566   \box_clear_new:N \l_@@_cell_box
1567   \normalbaselines

```

If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```

1568   \bool_if:NT \l_@@_small_bool
1569   {

```

⁵cf. `\nicematrix@redefine@check@rerun`

⁶The option `small` of `nicematrix` changes (among others) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1570     \def \arraystretch { 0.47 }
1571     \dim_set:Nn \arraycolsep { 1.45 pt }

```

By default, `\@@_tuning_key_small:` is no-op.

```

1572     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1573 }

```

The boolean `\g_@@_create_cell_nodes_bool` corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the “cell nodes” will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```

1574     \bool_if:NT \g_@@_create_cell_nodes_bool
1575     {
1576         \tbl_put_right:Nn \@@_begin_of_row:
1577         {
1578             \pgfsys@markposition
1579             { \@@_env: - row - \int_use:N \c@iRow - base }
1580         }
1581         \socket_assign_plug:nm { nicematrix / create-cell-nodes } { active }
1582     }

```

The environment `{array}` (since version 2.6) uses internally the command `\ar@ialign` (and previously, it was `\ialign`). We change that command for several reasons. In particular, `\ar@ialign` sets `\everycr` to `{ }` and we *need* to change the value of `\everycr`.

```

1583     \bool_if:NT \c_@@_recent_array_bool
1584     {
1585         \bool_if:NF \c_@@_revtex_bool
1586         {
1587             \def \ar@ialign
1588             {
1589                 \bool_if:NT \c_@@_testphase_table_bool
1590                 \tbl_init_cell_data_for_table:
1591                 \@@_some_initialization:
1592                 \dim_zero:N \tabskip

```

After its first use, the definition of `\ar@ialign` will revert automatically to its default definition. With this programming, we will have, in the cells of the array, a clean version of `\ar@ialign`. We use `\cs_set_eq:Nc` instead of `\cs_set_eq:NN` in order to avoid a message when `explcheck` is used on `nicematrix.sty`.

```

1593         \cs_set_eq:Nc \ar@ialign { \@@_old_ar@ialign: }
1594         \halign
1595     }
1596 }
1597 }

```

The following part should be deleted when we will delete the boolean `\c_@@_recent_array_bool` (when we consider the version 2.6a of `array` is required). Moreover, `revtex4-2` modifies `array` and provides commands which are meant to be the standard version of `array` but, at the date of november 2024, these commands corresponds to the *old* version of `array`, that is to say without the `\ar@ialign`.

```

1598     {
1599         \def \ialign
1600         {
1601             \@@_some_initialization:
1602             \dim_zero:N \tabskip
1603             \cs_set_eq:NN \ialign \@@_old_ialign:
1604             \halign
1605         }
1606     }

```

It seems that there is a problem when `nicematrix` is used with in `revtex4-2` with the package `colortbl` loaded. The following code prevent that problem but it does *not* treat the actual problem! It's only a patch *ad hoc*.

That patch has been added in version 7.0x, 2024-11-27 (question by mail of Tamra Nebabu).

```

1607 \bool_if:NT \c_@@_revtex_bool
1608 {
1609   \IfPackageLoadedT { colortbl }
1610   { \cs_set_protected:Npn \CT@setup { } }
1611 }

```

We keep in memory the old versions of `\ldots`, `\cdots`, etc. only because we use them inside `\phantom` commands in order that the new commands `\Ldots`, `\Cdots`, etc. give the same spacing (except when the option `nullify-dots` is used).

```

1612 \cs_set_eq:NN \@@_old_ldots: \ldots
1613 \cs_set_eq:NN \@@_old_cdots: \cdots
1614 \cs_set_eq:NN \@@_old_vdots: \vdots
1615 \cs_set_eq:NN \@@_old_ddots: \ddots
1616 \cs_set_eq:NN \@@_old_iddots: \iddots
1617 \bool_if:NTF \l_@@_standard_cline_bool
1618 { \cs_set_eq:NN \cline \@@_standard_cline: }
1619 { \cs_set_eq:NN \cline \@@_cline: }
1620 \cs_set_eq:NN \Ldots \@@_Ldots:
1621 \cs_set_eq:NN \Cdots \@@_Cdots:
1622 \cs_set_eq:NN \Vdots \@@_Vdots:
1623 \cs_set_eq:NN \Ddots \@@_Ddots:
1624 \cs_set_eq:NN \Iddots \@@_Iddots:
1625 \cs_set_eq:NN \Hline \@@_Hline:
1626 \cs_set_eq:NN \Hspace \@@_Hspace:
1627 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1628 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1629 \cs_set_eq:NN \Block \@@_Block:
1630 \cs_set_eq:NN \rotate \@@_rotate:
1631 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1632 \cs_set_eq:NN \dotfill \@@_dotfill:
1633 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1634 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1635 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1636 \cs_set_eq:NN \TopRule \@@_TopRule
1637 \cs_set_eq:NN \MidRule \@@_MidRule
1638 \cs_set_eq:NN \BottomRule \@@_BottomRule
1639 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1640 \cs_set_eq:NN \Hbrace \@@_Hbrace
1641 \cs_set_eq:NN \Vbrace \@@_Vbrace
1642 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1643 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1644 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1645 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1646 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1647 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1648 \int_compare:nNnT { \l_@@_first_row_int } > { \c_zero_int }
1649 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1650 \int_compare:nNnT { \l_@@_last_row_int } < { \c_zero_int }
1651 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1652 \bool_if:NT \l_@@_renew_dots_bool { \@@_renew_dots: }

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition. A `\hook_gremove_code:nn` will be put in `\@@_after_array:.`

```

1653 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1654 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1655 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn: }
1656 \@@_revert_colortbl:

```

If there is one or several commands `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the aux file at a previous run. We use that information to start counting the tabular notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

1657 \tl_if_exist:NT \l_@@_note_in_caption_tl
1658 {
1659   \tl_if_empty:NF \l_@@_note_in_caption_tl
1660   {
1661     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1662     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1663   }
1664 }

```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{n}{...}{...}` with $n > 1$ is issued. In `\g_@@_multicolumn_sizes_seq`, the “sizes” (that is to say the values of n) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```

1665 \seq_gclear:N \g_@@_multicolumn_cells_seq
1666 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

The counter `\c@iRow` will be used to count the rows of the array (its incrementation will be in the first cell of the row).

```

1667 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1668 \int_gzero_new:N \g_@@_row_total_int

```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_cell_begin:` executed at the beginning of each cell.

```

1669 \int_gzero_new:N \g_@@_col_total_int
1670 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1671 \bool_gset_false:N \g_@@_last_col_found_bool

```

During the construction of the array, the instructions `\Cdots`, `\Ldots`, etc. will be written in token lists `\g_@@_Cdots_lines_tl`, etc. which will be executed after the construction of the array.

```

1672 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1673 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1674 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1675 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1676 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1677 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1678 \tl_gclear:N \g_nicematrix_code_before_tl
1679 \tl_gclear:N \g_@@_pre_code_before_tl
1680 }

```

This is the end of `\@@_pre_array_ii:`.

The command `\@@_pre_array:` will be executed after analyse of the keys of the environment.

```

1681 \cs_new_protected:Npn \@@_pre_array:
1682 {
1683   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1684   \int_gzero_new:N \c@iRow
1685   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1686   \int_gzero_new:N \c@jCol

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-column` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it’s possible only after the first compilation).

```

1687 \int_compare:nNnT { \l_@@_last_row_int } = { -1 }

```



```

1688     {
1689       \bool_set_true:N \l_@@_last_row_without_value_bool
1690       \bool_if:NT \g_@@_aux_found_bool
1691       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
1692     }
1693   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
1694   {
1695     \bool_if:NT \g_@@_aux_found_bool
1696     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
1697   }

```

If there is an exterior row, we patch a command used in `\@@_cell_begin:` in order to keep track of some dimensions needed to the construction of that “last row”.

```

1698   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
1699   {
1700     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1701     {
1702       \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
1703       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1704       \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
1705       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1706     }
1707   }

1708   \seq_gclear:N \g_@@_cols_vlism_seq
1709   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1710   \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }

```

The value of `\g_@@_pos_of_blocks_seq` has been written on the aux file and loaded before the (potential) execution of the `\CodeBefore`. Now, we clear that variable because it will be reconstructed during the creation of the array.

```

1711   \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1712   \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1713   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1714   \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

The command `\create_row_node:` will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avoid warnings or, maybe, errors). That’s why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```

1715   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

The code in `\@@_pre_array_ii:` is used only here.

```

1716   \@@_pre_array_ii:

```

The array will be composed in a box (named `\l_@@_the_array_box`) because we have to do manipulations concerning the potential exterior rows.

```

1717   \box_clear_new:N \l_@@_the_array_box

```

We compute the width of both delimiters. We remind that, when the environment `{NiceArray}` is used, it's possible to specify the delimiters in the preamble (eg `[ccc]`).

```

1718 \dim_zero_new:N \l_@@_left_delim_dim
1719 \dim_zero_new:N \l_@@_right_delim_dim
1720 \bool_if:NTF \g_@@_delims_bool
1721 {

```

The command `\bBigg@` is a command of `amsmath`.

```

1722 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1723 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1724 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1725 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1726 }
1727 {
1728 \dim_gset:Nn \l_@@_left_delim_dim
1729 { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
1730 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1731 }

```

Here is the beginning of the box which will contain the array. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the second part of the environment (and the closing `\c_math_toggle_token` also).

```

1732 \hbox_set:Nw \l_@@_the_array_box
1733 \skip_horizontal:N \l_@@_left_margin_dim
1734 \skip_horizontal:N \l_@@_extra_left_margin_dim
1735 \bool_if:NT \c_@@_recent_array_bool
1736 { \UseTaggingSocket { tbl / hmode / begin } }

```

The following code is a workaround to specify to the tagging system that the following code is *fake math* (it raises `\l__math_fakemath_bool` in recent versions of LaTeX).

```

1737 \m@th
1738 \c_math_toggle_token
1739 \bool_if:NTF \l_@@_light_syntax_bool
1740 { \use:c { @@-light-syntax } }
1741 { \use:c { @@-normal-syntax } }
1742 }

```

The following command `\@@_CodeBefore_Body:w` will be used when the keyword `\CodeBefore` is present at the beginning of the environment.

```

1743 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1744 {
1745 \tl_set:Nn \l_tmpa_tl { #1 }
1746 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1747 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1748 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1749 \bool_set_true:N \l_@@_code_before_bool

```

We go on with `\@@_pre_array:` which will (among other) execute the `\CodeBefore` (specified in the key `code-before` or after the keyword `\CodeBefore`). By definition, the `\CodeBefore` must be executed before the body of the array...

```

1750 \@@_pre_array:
1751 }

```

9 The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed (that command will be used only once and is present alone only for legibility).

```

1752 \cs_new_protected:Npn \@@_pre_code_before:
1753 {

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1754     \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq { 2 } }
1755     \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq { 5 } }
1756     \int_set:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
1757     \int_set:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }

```

Now, we will create all the `col` nodes and `row` nodes with the information written in the `aux` file. You use the technique described in the page 1247 of `pgfmanual.pdf`, version 3.1.10.

```

1758     \pgfsys@markposition { \@@_env: - position }
1759     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1760     \pgfpicture
1761     \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1762     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int + 1 }
1763     {
1764         \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1765         \pgfcoordinate { \@@_env: - row - ##1 }
1766         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1767     }

```

Now, the recreation of the `col` nodes.

```

1768     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int + 1 }
1769     {
1770         \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1771         \pgfcoordinate { \@@_env: - col - ##1 }
1772         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1773     }

```

Now, you recreate the diagonal nodes by using the `row` nodes and the `col` nodes.

```

1774     \@@_create_diag_nodes:

```

Now, the creation of the `cell` nodes (`i-j`), and, maybe also the “medium nodes” and the “large nodes”.

```

1775     \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
1776     \endpgfpicture

```

Now, the recreation of the nodes of the blocks *which have a name*.

```

1777     \@@_create_blocks_nodes:
1778     \IfPackageLoadedT { tikz }
1779     {
1780         \tikzset
1781         {
1782             every-picture / .style =
1783             { overlay , name~prefix = \@@_env: - }
1784         }
1785     }
1786     \cs_set_eq:NN \cellcolor \@@_cellcolor
1787     \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1788     \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1789     \cs_set_eq:NN \rowcolor \@@_rowcolor
1790     \cs_set_eq:NN \rowcolors \@@_rowcolors
1791     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1792     \cs_set_eq:NN \arraycolor \@@_arraycolor
1793     \cs_set_eq:NN \columncolor \@@_columncolor
1794     \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1795     \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1796     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1797     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1798     \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1799   \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1800 }

```

```

1801 \cs_new_protected:Npn \@@_exec_code_before:
1802 {

```

We mark the cells which are in the (empty) corners because those cells must not be colored. We should try to find a way to detected whether we actually have coloring instructions to execute...

```

1803   \clist_map_inline:Nn \l_@@_corners_cells_clist
1804     { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1805   \seq_gclear_new:N \g_@@_colors_seq

```

The sequence `\g_@@_colors_seq` will always contain as first element the special color `nocolor`: when that color is used, no color will be applied in the corresponding cells by the other coloring commands of `nicematrix`.

```

1806   \@@_add_to_colors_seq:nn { { nocolor } } { }
1807   \bool_gset_false:N \g_@@_create_cell_nodes_bool
1808   \group_begin:

```

We compose the `\CodeBefore` in math mode in order to nullify the spaces put by the user between instructions in the `\CodeBefore`.

```

1809   \if_mode_math:
1810     \@@_exec_code_before_i:
1811   \else:
1812     \c_math_toggle_token
1813     \@@_exec_code_before_i:
1814     \c_math_toggle_token
1815   \fi:
1816   \group_end:
1817 }

```

The following code is a security for the case the user has used `babel` with the option `spanish`: in that case, the characters `<` (de code ASCII 60) and `>` are activated and Tikz is not able to solve the problem (even with the Tikz library `babel`).

```

1818 \cs_new_protected:Npn \@@_exec_code_before_i:
1819 {
1820   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1821     { \@@_rescan_for_spanish:N \l_@@_code_before_tl }

```

Here is the `\CodeBefore`. The construction is a bit complicated because `\g_@@_pre_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\g_@@_pre_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we use a `\q_stop`: it will be used to discard the rest of `\g_@@_pre_code_before_tl`.

```

1822   \exp_last_unbraced:No \@@_CodeBefore_keys:
1823   \g_@@_pre_code_before_tl

```

Now, all the cells which are specified to be colored by instructions in the `\CodeBefore` will actually be colored. It's a two-stages mechanism because we want to draw all the cells with the same color at the same time to absolutely avoid thin white lines in some PDF viewers.

```

1824   \@@_actually_color:
1825   \l_@@_code_before_tl
1826   \q_stop
1827 }

```

```

1828 \keys_define:nn { nicematrix / CodeBefore }
1829 {
1830   create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
1831   create-cell-nodes .default:n = true ,
1832   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1833   sub-matrix .value_required:n = true ,

```

```

1834     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1835     delimiters / color .value_required:n = true ,
1836     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1837   }

1838 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1839 {
1840   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1841   \@@_CodeBefore:w
1842 }

```

We have extracted the options of the keyword `\CodeBefore` in order to see whether the key `create-cell-nodes` has been used. Now, you can execute the rest of the `\CodeBefore`, excepted, of course, if we are in the first compilation.

```

1843 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1844 {
1845   \bool_if:NT \g_@@_aux_found_bool
1846   {
1847     \@@_pre_code_before:
1848     \legacy_if:nF { measuring@ } { #1 }
1849   }
1850 }

```

By default, if the user uses the `\CodeBefore`, only the `col` nodes, `row` nodes and `diag` nodes are available in that `\CodeBefore`. With the key `create-cell-nodes`, the cell nodes, that is to say the nodes of the form $(i-j)$ (but not the extra nodes) are also available because those nodes also are recreated and that recreation is done by the following command.

```

1851 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1852 {
1853   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
1854   {
1855     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1856     \pgfcoordinate { \@@_env: - row - ##1 - base }
1857     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1858     \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
1859     {
1860       \cs_if_exist:cT
1861       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1862       {
1863         \pgfsys@getposition
1864         { \@@_env: - ##1 - #####1 - NW }
1865         \@@_node_position:
1866         \pgfsys@getposition
1867         { \@@_env: - ##1 - #####1 - SE }
1868         \@@_node_position_i:
1869         \@@_pgf_rect_node:nnn
1870         { \@@_env: - ##1 - #####1 }
1871         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1872         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1873       }
1874     }
1875   }
1876   \@@_create_extra_nodes:
1877   \@@_create_aliases_last:
1878 }

1879 \cs_new_protected:Npn \@@_create_aliases_last:
1880 {
1881   \int_step_inline:nn { \c@iRow }
1882   {
1883     \pgfnodealias
1884     { \@@_env: - ##1 - last }
1885     { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

1886     }
1887 \int_step_inline:nn { \c@jCol }
1888   {
1889     \pgfnodealias
1890     { \@@_env: - last - ##1 }
1891     { \@@_env: - \int_use:N \c@iRow - ##1 }
1892   }
1893 \pgfnodealias % added 2025-04-05
1894   { \@@_env: - last - last }
1895   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1896 }

```

```

1897 \cs_new_protected:Npn \@@_create_blocks_nodes:
1898   {
1899     \pgfpicture
1900     \pgf@relevantforpicturesizefalse
1901     \pgfrememberpicturepositiononpagetrue
1902     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1903     { \@@_create_one_block_node:nnnnn ##1 }
1904     \endpgfpicture
1905   }

```

The following command is called `\@@_create_one_block_node:nnnnn` but, in fact, it creates a node only if the last argument (#5) which is the name of the block, is not empty.⁷

```

1906 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1907   {
1908     \tl_if_empty:nF { #5 }
1909     {
1910       \@@_qpoint:n { col - #2 }
1911       \dim_set_eq:NN \l_tmpa_dim \pgf@x
1912       \@@_qpoint:n { #1 }
1913       \dim_set_eq:NN \l_tmpb_dim \pgf@y
1914       \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1915       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1916       \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1917       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1918       \@@_pgf_rect_node:nnnnn
1919       { \@@_env: - #5 }
1920       { \dim_use:N \l_tmpa_dim }
1921       { \dim_use:N \l_tmpb_dim }
1922       { \dim_use:N \l_@@_tmpc_dim }
1923       { \dim_use:N \l_@@_tmpd_dim }
1924     }
1925   }

```

```

1926 \cs_new_protected:Npn \@@_patch_for_revtext:
1927   {
1928     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1929     \cs_set_eq:NN \@array \@array@array
1930     \cs_set_eq:NN \@tabular \@tabular@array
1931     \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1932     \cs_set_eq:NN \array \array@array
1933     \cs_set_eq:NN \endarray \endarray@array
1934     \cs_set:Npn \endtabular { \endarray $\egroup} % $
1935     \cs_set_eq:NN \@mkpream \@mkpream@array
1936     \cs_set_eq:NN \@classx \@classx@array
1937     \cs_set_eq:NN \insert@column \insert@column@array
1938     \cs_set_eq:NN \@arraycr \@arraycr@array

```

⁷Moreover, there is also in the list `\g_@@_pos_of_blocks_seq` the positions of the dotted lines (created by `\Cdots`, etc.) and, for these entries, there is, of course, no name (the fifth component is empty).

```

1939   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1940   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1941 }

```

10 The environment `{NiceArrayWithDelims}`

```

1942 \NewDocumentEnvironment { NiceArrayWithDelims }
1943 { m m 0 { } m ! 0 { } t \CodeBefore }
1944 {
1945   \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
1946   \@@_provide_pgfsyspdfmark:
1947   \bool_if:NT \g_@@_footnote_bool { \savenotes }

```

The aim of the following `\bgroup` (the corresponding `\egroup` is, of course, at the end of the environment) is to be able to put an exponent to a matrix in a mathematical formula.

```

1948   \bgroup

1949   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1950   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1951   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1952   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1953   \int_gzero:N \g_@@_block_box_int
1954   \dim_gzero:N \g_@@_width_last_col_dim
1955   \dim_gzero:N \g_@@_width_first_col_dim
1956   \bool_gset_false:N \g_@@_row_of_col_done_bool
1957   \str_if_empty:NT \g_@@_name_env_str
1958     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1959   \bool_if:NTF \l_@@_tabular_bool
1960     { \mode_leave_vertical: }
1961     { \@@_test_if_math_mode: }
1962   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in-env } }
1963   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁸. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by `colortbl`. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```

1964   \cs_gset_eq:cN { @@_old_CT@arc@ } \CT@arc@

```

We deactivate Tikz externalization because we will use PGF pictures with the options `overlay` and `remember picture` (or equivalent forms). We deactivate with `\tikzexternaldisable` and not with `\tikzset{external/export=false}` which is *not* equivalent.

```

1965   \cs_if_exist:NT \tikz@library@external@loaded
1966   {
1967     \tikzexternaldisable
1968     \cs_if_exist:NT \ifstandalone
1969       { \tikzset { external / optimize = false } }
1970   }

```

We increment the counter `\g_@@_env_int` which counts the environments of the package.

```

1971   \int_gincr:N \g_@@_env_int
1972   \bool_if:NF \l_@@_block_auto_columns_width_bool
1973     { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

⁸e.g. `\color[rgb]{0.5,0.5,0}`

The sequence `\g_@@_blocks_seq` will contain the characteristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks.

```
1974 \seq_gclear:N \g_@@_blocks_seq
1975 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

In fact, the sequence `\g_@@_pos_of_blocks_seq` will also contain the positions of the cells with a `\diagbox` and the `\multicolumn`.

```
1976 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1977 \seq_gclear:N \g_@@_pos_of_xdots_seq
1978 \tl_gclear_new:N \g_@@_code_before_tl
1979 \tl_gclear:N \g_@@_row_style_tl
```

We load all the information written in the aux file during previous compilations corresponding to the current environment.

```
1980 \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1981 {
1982   \bool_gset_true:N \g_@@_aux_found_bool
1983   \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
1984 }
1985 { \bool_gset_false:N \g_@@_aux_found_bool }
```

Now, we prepare the token list for the instructions that we will have to write on the aux file at the end of the environment.

```
1986 \tl_gclear:N \g_@@_aux_tl
1987 \tl_if_empty:NF \g_@@_code_before_tl
1988 {
1989   \bool_set_true:N \l_@@_code_before_bool
1990   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1991 }
1992 \tl_if_empty:NF \g_@@_pre_code_before_tl
1993 { \bool_set_true:N \l_@@_code_before_bool }
```

The set of keys is not exactly the same for `{NiceArray}` and for the variants of `{NiceArray}` (`{pNiceArray}`, `{bNiceArray}`, etc.) because, for `{NiceArray}`, we have the options `t`, `c`, `b` and `baseline`.

```
1994 \bool_if:NTF \g_@@_delims_bool
1995 { \keys_set:nn { nicematrix / pNiceArray } }
1996 { \keys_set:nn { nicematrix / NiceArray } }
1997 { #3 , #5 }

1998 \@@_set_CTarC:o \l_@@_rules_color_tl % noqa: w302
```

The argument `#6` is the last argument of `{NiceArrayWithDelims}`. With that argument of type “`t \CodeBefore`”, we test whether there is the keyword `\CodeBefore` at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword `\CodeBefore` and the (other) keyword `\Body`. It's the job that will do the command `\@@_CodeBefore_Body:w`. After that job, the command `\@@_CodeBefore_Body:w` will go on with `\@@_pre_array:`.

```
1999 \bool_if:nTF { #6 } { \@@_CodeBefore_Body:w } { \@@_pre_array: }
2000 }
```

Now, the second part of the environment `{NiceArrayWithDelims}`.

```
2001 {
2002   \bool_if:NTF \l_@@_light_syntax_bool
2003   { \use:c { end @@-light-syntax } }
2004   { \use:c { end @@-normal-syntax } }
2005   \c_math_toggle_token
2006   \skip_horizontal:N \l_@@_right_margin_dim
2007   \skip_horizontal:N \l_@@_extra_right_margin_dim
2008   \hbox_set_end:
2009   \bool_if:NT \c_@@_recent_array_bool
2010   { \UseTaggingSocket { tbl / hmode / end } }
```

End of the construction of the array (in the box `\l_@@_the_array_box`).

If the user has used the key `width` without any column `X`, we raise an error.

```

2011   \bool_if:NT \l_@@_width_used_bool
2012   {
2013     \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
2014     { \@@_error_or_warning:n { width-without-X~columns } }
2015   }

```

Now, if there is at least one `X`-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `l_@@_X_columns_dim` will be the width of a column of weight 1.0. For a `X`-column of weight x , the width will be `l_@@_X_columns_dim` multiplied by x .

```

2016   \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
2017   { \@@_compute_width_X: }

```

If the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the actual number of rows of the array).

```

2018   \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
2019   {
2020     \bool_if:NF \l_@@_last_row_without_value_bool
2021     {
2022       \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
2023       {
2024         \@@_error:n { Wrong~last-row }
2025         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2026       }
2027     }
2028   }

```

Now, the definition of `\c@jCol` and `\g_@@_col_total_int` changes: `\c@jCol` will be the number of columns without the “last column”; `\g_@@_col_total_int` will be the number of columns with this “last column”.⁹

```

2029   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2030   \bool_if:NTF \g_@@_last_col_found_bool
2031   { \int_gdecr:N \c@jCol }
2032   {
2033     \int_compare:nNnT { \l_@@_last_col_int } > { -1 }
2034     { \@@_error:n { last~col~not~used } }
2035   }

```

We fix also the value of `\c@iRow` and `\g_@@_row_total_int` with the same principle.

```

2036   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2037   \int_compare:nNnT { \l_@@_last_row_int } > { -1 }
2038   { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 92).

```

2039   \int_if_zero:nT { \l_@@_first_col_int }
2040   { \skip_horizontal:N \g_@@_width_first_col_dim }

```

The construction of the real box is different whether we have delimiters to put.

```

2041   \bool_if:nTF { ! \g_@@_delims_bool }
2042   {
2043     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2044     { \@@_use_arraybox_with_notes_c: }
2045     {
2046       \str_if_eq:eeTF \l_@@_baseline_tl { b }
2047       { \@@_use_arraybox_with_notes_b: }
2048       { \@@_use_arraybox_with_notes: }
2049     }
2050   }

```

⁹We remind that the potential “first column” (exterior) has the number 0.

Now, in the case of an environment with delimiters. We compute `\l_tmpa_dim` which is the total height of the “first row” above the array (when the key `first-row` is used).

```

2051     {
2052         \int_if_zero:nTF { \l_@@_first_row_int }
2053         {
2054             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2055             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2056         }
2057         { \dim_zero:N \l_tmpa_dim }

```

We compute `\l_tmpb_dim` which is the total height of the “last row” below the array (when the key `last-row` is used). A value of `-2` for `\l_@@_last_row_int` means that there is no “last row”.¹⁰

```

2058         \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
2059         {
2060             \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2061             \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2062         }
2063         { \dim_zero:N \l_tmpb_dim }
2064     \hbox_set:Nn \l_tmpa_box
2065     {
2066         \m@th
2067         \c_math_toggle_token
2068         \@@_color:o \l_@@_delimiters_color_tl
2069         \exp_after:wN \left \g_@@_left_delim_tl
2070         \vcenter
2071         {

```

We take into account the “first row” (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here.

```

2072             \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
2073             \hbox
2074             {
2075                 \bool_if:NTF \l_@@_tabular_bool
2076                 { \skip_horizontal:n { - \tabcolsep } }
2077                 { \skip_horizontal:n { - \arraycolsep } }
2078                 \@@_use_arraybox_with_notes_c:
2079                 \bool_if:NTF \l_@@_tabular_bool
2080                 { \skip_horizontal:n { - \tabcolsep } }
2081                 { \skip_horizontal:n { - \arraycolsep } }
2082             }

```

We take into account the “last row” (we have previously computed its total height in `\l_tmpb_dim`).

```

2083             \skip_vertical:n { - \l_tmpb_dim + \arrayrulewidth }
2084         }
2085         \exp_after:wN \right \g_@@_right_delim_tl
2086         \c_math_toggle_token
2087     }

```

Now, the box `\l_tmpa_box` is created with the correct delimiters.

We will put the box in the TeX flow. However, we have a small work to do when the option `delimiters/max-width` is used.

```

2088     \bool_if:NTF \l_@@_delimiters_max_width_bool
2089     {
2090         \@@_put_box_in_flow_bis:nn
2091         { \g_@@_left_delim_tl }
2092         { \g_@@_right_delim_tl }
2093     }
2094     \@@_put_box_in_flow:
2095 }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 93).

¹⁰A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```

2096 \bool_if:NT \g_@@_last_col_found_bool
2097   { \skip_horizontal:N \g_@@_width_last_col_dim }
2098 \bool_if:NT \l_@@_preamble_bool
2099   {
2100     \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
2101     { \@@_err_columns_not_used: }
2102   }
2103 \@@_after_array:

```

The aim of the following `\egroup` (the corresponding `\bgroup` is, of course, at the beginning of the environment) is to be able to put an exponent to a matrix in a mathematical formula.

```

2104 \egroup

```

We write on the aux file all the information corresponding to the current environment.

```

2105 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2106 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2107 \iow_now:Ne \@mainaux
2108   {
2109     \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
2110     \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
2111     { \exp_not:o \g_@@_aux_tl }
2112   }
2113 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2114 \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
2115 }

```

This is the end of the environment `{NiceArrayWithDelims}`.

```

2116 \cs_new_protected:Npn \@@_err_columns_not_used:
2117   {
2118     \@@_warning:n { columns~not~used }
2119     \cs_gset:Npn \@@_err_columns_not_used: { }
2120   }

```

The following command will be used only once. We have written that command for legibility. If there is at least one X-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `l_@@_X_columns_dim` will be the width of a column of weight 1.0. For a X-column of weight x , the width will be `l_@@_X_columns_dim` multiplied by x .

```

2121 \cs_new_protected:Npn \@@_compute_width_X:
2122   {
2123     \tl_gput_right:Ne \g_@@_aux_tl
2124     {
2125       \bool_set_true:N \l_@@_X_columns_aux_bool
2126       \dim_set:Nn \l_@@_X_columns_dim
2127     }

```

The flag `g_@@_V_of_X_bool` is raised when there is at least in the tabular a column of type X using the key V. In that case, the width of the X column may be considered as correct even though the tabular has not (of course) a width equal to `l_@@_width_dim`

```

2128     \bool_lazy_and:nnTF
2129       { \g_@@_V_of_X_bool }
2130       { \l_@@_X_columns_aux_bool }
2131       { \dim_use:N \l_@@_X_columns_dim }
2132     {
2133       \dim_compare:nNnTF
2134         {
2135           \dim_abs:n
2136             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2137         }
2138         <
2139         { 0.001 pt }
2140         { \dim_use:N \l_@@_X_columns_dim }

```

```

2141         {
2142             \dim_eval:n
2143             {
2144                 \l_@@_X_columns_dim
2145                 +
2146                 \fp_to_dim:n
2147                 {
2148                     (
2149                         \dim_eval:n
2150                         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2151                     )
2152                     / \fp_use:N \g_@@_total_X_weight_fp
2153                 }
2154             }
2155         }
2156     }
2157 }
2158 }
2159 }
2160 }

```

11 Construction of the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble which will be given to `{array}` (of the package `array`).

The preamble given by the final user is stored in `\g_@@_user_preamble_tl`. The modified version will be stored in `\g_@@_array_preamble_tl`.

```

2161 \cs_new_protected:Npn \@@_transform_preamble:
2162 {
2163     \@@_transform_preamble_i:
2164     \@@_transform_preamble_ii:
2165 }
2166 \cs_new_protected:Npn \@@_transform_preamble_i:
2167 {
2168     \int_gzero:N \c@jCol

```

The sequence `\g_@@_cols_vlism_seq` will contain the numbers of the columns where you will to have to draw vertical lines in the potential sub-matrices (hence the name `vlism`).

```

2169     \seq_gclear:N \g_@@_cols_vlism_seq

```

`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble provided by the final user.

```

2170     \bool_gset_false:N \g_tmpb_bool

```

The following sequence will store the arguments of the successive `>` in the preamble.

```

2171     \tl_gclear_new:N \g_@@_pre_cell_tl

```

The counter `\l_tmpa_int` will count the number of consecutive occurrences of the symbol `|`.

```

2172     \int_zero:N \l_tmpa_int
2173     \tl_gclear:N \g_@@_array_preamble_tl
2174     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2175     {
2176         \tl_gset:Nn \g_@@_array_preamble_tl
2177         { ! { \skip_horizontal:N \arrayrulewidth } }
2178     }
2179     {
2180         \clist_if_in:NnT \l_@@_vlines_clist 1
2181         {
2182             \tl_gset:Nn \g_@@_array_preamble_tl

```

```

2183         { ! { \skip_horizontal:N \arrayrulewidth } }
2184     }
2185 }

```

Now, we actually make the preamble (which will be given to `{array}`). It will be stored in `\g_@@_array_preamble_tl`.

```

2186     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
2187     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2188     \@@_replace_columncolor:
2189 }

2190 \cs_new_protected:Npn \@@_transform_preamble_ii:
2191 {

```

If there were delimiters at the beginning or at the end of the preamble, the environment `{NiceArray}` is transformed into an environment `{xNiceMatrix}`.

```

2192     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2193     {
2194         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2195         { \bool_gset_true:N \g_@@_delims_bool }
2196     }
2197     { \bool_gset_true:N \g_@@_delims_bool }

```

We want to remind whether there is a specifier `|` at the end of the preamble.

```

2198     \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }

```

We complete the preamble with the potential “exterior columns” (on both sides).

```

2199     \int_if_zero:nTF { \l_@@_first_col_int }
2200     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2201     {
2202         \bool_if:NF \g_@@_delims_bool
2203         {
2204             \bool_if:NF \l_@@_tabular_bool
2205             {
2206                 \clist_if_empty:NT \l_@@_vlines_clist
2207                 {
2208                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2209                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2210                 }
2211             }
2212         }
2213     }
2214     \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
2215     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2216     {
2217         \bool_if:NF \g_@@_delims_bool
2218         {
2219             \bool_if:NF \l_@@_tabular_bool
2220             {
2221                 \clist_if_empty:NT \l_@@_vlines_clist
2222                 {
2223                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2224                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2225                 }
2226             }
2227         }
2228     }

```

We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it's not possible to do that in `{NiceTabular*}` (we control that with the value of `\l_@@_tabular_width_dim`).

```
2229 \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
2230 {
```

If the tagging of the tabulars is done (part of the Tagging Project), you don't activate that mechanism because it would create a dummy column of tagged empty cells.

```
2231 \bool_if:NF \c_@@_testphase_table_bool
2232 {
2233 \tl_gput_right:Nn \g_@@_array_preamble_tl
2234 { > { \@@_error_too_much_cols: } 1 }
2235 }
2236 }
2237 }
```

The preamble provided by the final user will be read by a finite automata. The following function `\@@_rec_preamble:n` will read that preamble (usually letter by letter) in a recursive way (hence the name of that function). in the preamble.

```
2238 \cs_new_protected:Npn \@@_rec_preamble:n #1
2239 {
```

For the majority of the letters, we will trigger the corresponding action by calling directly a function in the main hashtable of TeX (thanks to the mechanism `\csname...\endcsname`. Be careful: all these functions take in as first argument the letter (or token) itself.¹¹

```
2240 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 : }
2241 { \use:c { @@ _ \token_to_str:N #1 : } { #1 } }
2242 {
```

Now, the columns defined by `\newcolumntype` of `array`.

```
2243 \cs_if_exist:cTF { NC @ find @ #1 }
2244 {
2245 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2246 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2247 }
2248 {
2249 \str_if_eq:nnTF { #1 } { S }
2250 { \@@_fatal:n { unknown~column~type~S } }
2251 { \@@_fatal:nn { unknown~column~type } { #1 } }
2252 }
2253 }
2254 }
```

For `c`, `l` and `r`

```
2255 \cs_new_protected:Npn \@@_c: #1
2256 {
2257 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2258 \tl_gclear:N \g_@@_pre_cell_tl
2259 \tl_gput_right:Nn \g_@@_array_preamble_tl
2260 { > \@@_cell_begin: c < \@@_cell_end: }
```

We increment the counter of columns and then we test for the presence of a `<`.

```
2261 \int_gincr:N \c_jCol
2262 \@@_rec_preamble_after_col:n
2263 }
```

¹¹We do that because it's an easy way to insert the letter at some places in the code that we will add to `\g_@@_array_preamble_tl`.

```

2264 \cs_new_protected:Npn \@@_l: #1
2265 {
2266   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2267   \tl_gclear:N \g_@@_pre_cell_tl
2268   \tl_gput_right:Nn \g_@@_array_preamble_tl
2269   {
2270     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2271     l
2272     < \@@_cell_end:
2273   }
2274   \int_gincr:N \c@jCol
2275   \@@_rec_preamble_after_col:n
2276 }

2277 \cs_new_protected:Npn \@@_r: #1
2278 {
2279   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2280   \tl_gclear:N \g_@@_pre_cell_tl
2281   \tl_gput_right:Nn \g_@@_array_preamble_tl
2282   {
2283     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2284     r
2285     < \@@_cell_end:
2286   }
2287   \int_gincr:N \c@jCol
2288   \@@_rec_preamble_after_col:n
2289 }

```

For ! and @

```

2290 \cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
2291 {
2292   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2293   \@@_rec_preamble:n
2294 }
2295 \cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }

```

For |

```

2296 \cs_new_protected:cpn { @@ _ | : } #1
2297 {
\l_tmpa_int is the number of successive occurrences of |
2298   \int_incr:N \l_tmpa_int
2299   \@@_make_preamble_i_i:n
2300 }

2301 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2302 {

```

Here, we can't use \str_if_eq:eeTF.

```

2303   \str_if_eq:nnTF { #1 } { | }
2304   { \use:c { @@ _ | : } | }
2305   { \@@_make_preamble_i_ii:nn { } #1 }
2306 }

2307 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2308 {
2309   \str_if_eq:nnTF { #2 } { [ ]
2310     { \@@_make_preamble_i_ii:nw { #1 } [ ]
2311       { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2312     }
2313   \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2314   { \@@_make_preamble_i_ii:nn { #1 , #2 } }

```

```

2315 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2316 {
2317   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2318   \tl_gput_right:Ne \g_@@_array_preamble_tl
2319   {

```

Here, the command `\dim_use:N` is mandatory.

```

2320     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2321   }
2322   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2323   {
2324     \@@_vline:n
2325     {
2326       position = \int_eval:n { \c@jCol + 1 } ,
2327       multiplicity = \int_use:N \l_tmpa_int ,
2328       total-width = \dim_use:N \l_@@_rule_width_dim ,
2329       #2
2330     }

```

We don't have provided value for `start` nor for `end`, which means that the rule will cover (potentially) all the rows of the array.

```

2331   }
2332   \int_zero:N \l_tmpa_int
2333   \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
2334   \@@_rec_preamble:n #1
2335 }

```

```

2336 \cs_new_protected:cpn { @@_ > : } #1 #2
2337 {
2338   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2339   \@@_rec_preamble:n
2340 }
2341 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

The specifier `p` (and also the specifiers `m`, `b`, `V` and `X`) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys.

```

2342 \keys_define:nn { nicematrix / p-column }
2343 {
2344   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2345   r .value_forbidden:n = true ,
2346   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2347   c .value_forbidden:n = true ,
2348   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2349   l .value_forbidden:n = true ,
2350   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2351   S .value_forbidden:n = true ,
2352   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2353   p .value_forbidden:n = true ,
2354   t .meta:n = p ,
2355   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2356   m .value_forbidden:n = true ,
2357   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2358   b .value_forbidden:n = true
2359 }

```

For `p` but also `b` and `m`.

```

2360 \cs_new_protected:Npn \@@_p: #1
2361 {
2362   \str_set:Nn \l_@@_vpos_col_str { #1 }

```


Now, you look for a potential character [after the letter of the specifier (for the options).

```

2363   \@@_make_preamble_ii_i:n
2364   }
2365   \cs_set_eq:NN \@@_b: \@@_p:
2366   \cs_set_eq:NN \@@_m: \@@_p:
2367   \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2368   {
2369     \str_if_eq:nnTF { #1 } { [ ]
2370     { \@@_make_preamble_ii_ii:w [ ]
2371     { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2372     }
2373   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2374   { \@@_make_preamble_ii_iii:nn { #1 } }

```

#1 is the optional argument of the specifier (a list of *key-value* pairs).

#2 is the mandatory argument of the specifier: the width of the column.

```

2375   \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2376   {

```

The possible values of `\l_@@_hpos_col_str` are *j* (for *justified* which is the initial value), *l*, *c*, *r*, *L*, *C* and *R* (when the user has used the corresponding key in the optional argument of the specifier).

```

2377     \str_set:Nn \l_@@_hpos_col_str { j }
2378     \@@_keys_p_column:n { #1 }

```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2379     \setlength { \l_tmpa_dim } { #2 }
2380     \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2381   }
2382   \cs_new_protected:Npn \@@_keys_p_column:n #1
2383   { \keys_set_known:nnN { nicematrix / p-column } { #1 } \l_tmpa_tl }

```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`. The third is some code added at the beginning of the cell.

```

2384   \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2385   {

```

Here, `\expanded` would probably be slightly faster than `\use:e`

```

2386     \use:e
2387     {
2388       \@@_make_preamble_ii_vi:nnnnnnn
2389       { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2390       { #1 }
2391     }

```

The parameter `\l_@@_hpos_col_str` (as `\l_@@_vpos_col_str`) exists only during the construction of the preamble. During the composition of the array itself, you will have, in each cell, the parameter `\l_@@_hpos_cell_tl` which will provide the horizontal alignment of the column to which belongs the cell.

```

2392         \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2393         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2394         {

```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```

2395             \def \exp_not:N \l_@@_hpos_cell_tl
2396             { \str_lowercase:f { \l_@@_hpos_col_str } }
2397         }
2398     \IfPackageLoadedTF { ragged2e }
2399     {
2400         \str_case:on \l_@@_hpos_col_str
2401         {

```

The following `\exp_not:N` are mandatory.

```

2402         c { \exp_not:N \Centering }
2403         l { \exp_not:N \RaggedRight }
2404         r { \exp_not:N \RaggedLeft }
2405     }
2406 }
2407 {
2408     \str_case:on \l_@@_hpos_col_str
2409     {
2410         c { \exp_not:N \centering }
2411         l { \exp_not:N \raggedright }
2412         r { \exp_not:N \raggedleft }
2413     }
2414 }
2415 #3
2416 }
2417 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2418 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2419 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2420 { #2 }
2421 {
2422     \str_case:onF \l_@@_hpos_col_str
2423     {
2424         { j } { c }
2425         { si } { c }
2426     }

```

We use `\str_lowercase:n` to convert R to r, etc.

```

2427         { \str_lowercase:f \l_@@_hpos_col_str }
2428     }
2429 }

```

We increment the counter of columns, and then we test for the presence of a <.

```

2430     \int_gincr:N \c@jCol
2431     \@@_rec_preamble_after_col:n
2432 }

```

#1 is the optional argument of `{minipage}` (or `{varwidth}`): t or b. Indeed, for the columns of type m, we use the value b here because there is a special post-action in order to center vertically the box (see #4).

#2 is the width of the `{minipage}` (or `{varwidth}`), that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\raggedleft` or nothing). It's also possible to put in that #3 some code to fix the value of `\l_@@_hpos_cell_tl` which will be available in each cell of the column.

#4 is an extra-code which contains `\@@_center_cell_box:` (when the column is a m column) or nothing (in the other cases).

#5 is a code put just before the c (or r or l: see #8).

#6 is a code put just after the c (or r or l: see #8).

#7 is the type of environment: `minipage` or `varwidth`.

#8 is the letter c or r or l which is the basic specifier of column which is used *in fine*.

```

2433 \cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2434 {
2435     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2436     {
2437         \tl_gput_right:Nn \g_@@_array_preamble_tl
2438         { > \@@_test_if_empty_for_S: }
2439     }
2440     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2441     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2442     \tl_gclear:N \g_@@_pre_cell_tl
2443     \tl_gput_right:Nn \g_@@_array_preamble_tl
2444     {
2445         > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```

2446         \dim_set:Nn \l_@@_col_width_dim { #2 }
2447         \bool_if:NT \c_@@_testphase_table_bool
2448           { \tag_struct_begin:n { tag = Div } }
2449         \@@_cell_begin:

```

We use the form `\minipage–\endminipage (\varwidth–\endvarwidth)` for compatibility with `colcell` (2023-10-31).

```

2450         \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from `array.sty`.

```

2451         \everypar
2452           {
2453             \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2454             \everypar { }
2455           }
2456         \bool_if:NT \c_@@_testphase_table_bool { \tagpdfpara0n }

```

Now, the potential code for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\RaggedRight`, etc.).

```

2457         #3

```

The following code is to allow something like `\centering` in `\RowStyle`.

```

2458         \g_@@_row_style_tl
2459         \arraybackslash
2460         #5
2461       }
2462     #8
2463   < {
2464     #6

```

The following line has been taken from `array.sty`.

```

2465         \@finalstrut \@arstrutbox
2466         \use:c { end #7 }

```

If the letter in the preamble is `m`, `#4` will be equal to `\@@_center_cell_box:` (see just below).

```

2467         #4
2468         \@@_cell_end:
2469         \bool_if:NT \c_@@_testphase_table_bool { \tag_struct_end: }
2470       }
2471     }
2472   }

```

The cell always begins with `\ignorespaces` with `array` and that's why we retrieve that token.

```

2473 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2474 {

```

We open a special group with `\group_align_safe_begin:`. Thus, when `\peek_meaning:NTF` will read the `&` (when the cell is empty), that lecture won't trigger the end of the cell (since we are in a lower group...). If the end of cell was triggered, we would have other tokens in the TeX flow (and not `&`).

```

2475 \group_align_safe_begin:
2476 \peek_meaning:NTF &
2477 { \@@_the_cell_is_empty: }
2478 {
2479   \peek_meaning:NTF \\\
2480   { \@@_the_cell_is_empty: }
2481   {
2482     \peek_meaning:NTF \crcr
2483     \@@_the_cell_is_empty:
2484     \group_align_safe_end:
2485   }
2486 }
2487 }

```

```

2488 \cs_new_protected:Npn \@@_the_cell_is_empty:
2489 {
2490   \group_align_safe_end:
2491   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2492   {

```

Be careful: here, we can't merely use `\bool_gset_true: \g_@@_empty_cell_bool`, in particular because of the columns of type X.

```

2493     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2494     \skip_horizontal:N \l_@@_col_width_dim
2495   }
2496 }

2497 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2498 {
2499   \peek_meaning:NT \__siunitx_table_skip:n
2500   { \bool_gset_true:N \g_@@_empty_cell_bool }
2501 }

```

The following command will be used in m-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in `array`) that if the height of the cell is no more that the height of `\strutbox`, there is only one row.

```

2502 \cs_new_protected:Npn \@@_center_cell_box:
2503 {

```

By putting instructions in `\g_@@_cell_after_hook_tl`, we require a post-action of the box `\l_@@_cell_box`.

```

2504   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2505   {
2506     \dim_compare:nNnT
2507     { \box_ht:N \l_@@_cell_box }
2508     >

```

Previously, we had `\@arstrutbox` and not `\strutbox` in the following line but the code in `array` has changed in v 2.5g and we follow the change (see *array: Correctly identify single-line m-cells* in *LaTeX News 36*).

```

2509     { \box_ht:N \strutbox }
2510     {
2511       \hbox_set:Nn \l_@@_cell_box
2512       {
2513         \box_move_down:nn
2514         {
2515           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2516             + \baselineskip ) / 2
2517         }
2518         { \box_use:N \l_@@_cell_box }
2519       }
2520     }
2521   }
2522 }

```

For V (similar to the V of `varwidth`).

```

2523 \cs_new_protected:Npn \@@_V: #1 #2
2524 {
2525   \str_if_eq:nnTF { #1 } { [ ]
2526     { \@@_make_preamble_V_i:w [ ]
2527       { \@@_make_preamble_V_i:w [ ] { #2 } }
2528     }
2529   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2530     { \@@_make_preamble_V_ii:nn { #1 } }
2531   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2

```

```

2532 {
2533   \str_set:Nn \l_@@_vpos_col_str { p }
2534   \str_set:Nn \l_@@_hpos_col_str { j }
2535   \@@_keys_p_column:n { #1 }

```

We apply `\setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package `calc` (not loaded by `nicematrix`) which redefines the command `\setlength`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2536   \setlength { \l_tmpa_dim } { #2 }
2537   \IfPackageLoadedTF { varwidth }
2538     { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
2539     {
2540       \@@_error_or_warning:n { varwidth-not-loaded }
2541       \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
2542     }
2543 }

```

For `w` and `W`

```

2544 \cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
2545 \cs_new_protected:Npn \@@_W: { \@@_make_preamble_w:nnnn { \@@_special_W: } }

```

#1 is a special argument: empty for `w` and equal to `\@@_special_W:` for `W`;
#2 is the type of column (`w` or `W`);
#3 is the type of horizontal alignment (`c`, `l`, `r` or `s`);
#4 is the width of the column.

```

2546 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2547 {
2548   \str_if_eq:nnTF { #3 } { s }
2549     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2550     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2551 }

```

First, the case of an horizontal alignment equal to `s` (for *stretch*).

#1 is a special argument: empty for `w` and equal to `\@@_special_W:` for `W`;
#2 is the width of the column.

```

2552 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2553 {
2554   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2555   \tl_gclear:N \g_@@_pre_cell_tl
2556   \tl_gput_right:Nn \g_@@_array_preamble_tl
2557     {
2558       > {
2559         % We use |\setlength| in order to allow |\widthof| which is a command of \pkg{calc}
2560         % (when loaded \pkg{calc} redefines |\setlength|).
2561         % Of course, even if \pkg{calc} is not loaded, the following code will work with
2562         % the standard version of |\setlength|.
2563         \setlength { \l_@@_col_width_dim } { #2 }
2564         \@@_cell_begin:
2565         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2566       }
2567       c
2568       < {
2569         \@@_cell_end_for_w_s:
2570         #1
2571         \@@_adjust_size_box:
2572         \box_use_drop:N \l_@@_cell_box
2573       }
2574     }
2575   \int_gincr:N \c@jCol
2576   \@@_rec_preamble_after_col:n
2577 }

```

Then, the most important version, for the horizontal alignments types of c, l and r (and not s).

```

2578 \cs_new_protected:Npn \@@_make_preamble_w_ii:nmmn #1 #2 #3 #4
2579 {
2580   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2581   \tl_gcLEAR:N \g_@@_pre_cell_tl
2582   \tl_gput_right:Nn \g_@@_array_preamble_tl
2583   {
2584     > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

We use `\setlength` in order to allow `\widthof` which is a command of `calc` (when loaded `calc` redefines `\setlength`). Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2585     \setlength { \l_@@_col_width_dim } { #4 }
2586     \hbox_set:Nw \l_@@_cell_box
2587     \@@_cell_begin:
2588     \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
2589   }
2590   c
2591   < {
2592     \@@_cell_end:
2593     \hbox_set_end:
2594     #1
2595     \@@_adjust_size_box:
2596     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2597   }
2598 }

```

We increment the counter of columns and then we test for the presence of a <.

```

2599   \int_gincr:N \c@jCol
2600   \@@_rec_preamble_after_col:n
2601 }

2602 \cs_new_protected:Npn \@@_special_W:
2603 {
2604   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \l_@@_col_width_dim }
2605   { \@@_warning:n { W~warning } }
2606 }

```

For S (of `siunitx`).

```

2607 \cs_new_protected:Npn \@@_S: #1 #2
2608 {
2609   \str_if_eq:nnTF { #2 } { [ ] }
2610   { \@@_make_preamble_S:w [ ] }
2611   { \@@_make_preamble_S:w [ ] { #2 } }
2612 }

2613 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2614 { \@@_make_preamble_S_i:n { #1 } }

2615 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2616 {
2617   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2618   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2619   \tl_gcLEAR:N \g_@@_pre_cell_tl
2620   \tl_gput_right:Nn \g_@@_array_preamble_tl
2621   {
2622     > {

```

In the cells of a column of type S, we have to wrap the command `\@@_node_cell:` for the horizontal alignment of the content of the cell (`siunitx` has done a job but it's without effect since we have to put the content in a box for the PGF/TikZ node and that's why we have to do the job of horizontal alignment once again).

```

2623     \socket_assign_plug:n { nicematrix / siunitx-wrap } { active }
2624     \keys_set:n { siunitx } { #1 }
2625     \@@_cell_begin:
2626     \siunitx_cell_begin:w
2627   }
2628   c
2629   <
2630   {
2631     \siunitx_cell_end:

```

We want the value of `\l__siunitx_table_text_bool` available *after* `\@@_cell_end:` because we need it to know how to align our box after the construction of the PGF/TikZ node. That's why we use `\g_@@_cell_after_hook_tl` to reset the correct value of `\l__siunitx_table_text_bool` (of course, it will stay local within the cell of the underlying `\halign`).

```

2632     \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2633     {
2634       \bool_if:NTF \l__siunitx_table_text_bool
2635         { \bool_set_true:N }
2636         { \bool_set_false:N }
2637       \l__siunitx_table_text_bool
2638     }
2639     \@@_cell_end:
2640   }
2641 }

```

We increment the counter of columns and then we test for the presence of a `<`.

```

2642   \int_gincr:N \c@jCol
2643   \@@_rec_preamble_after_col:n
2644 }

```

For `(`, `[` and `\{`.

```

2645 \cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
2646 {
2647   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }

```

If we are before the column 1 and not in `{NiceArray}`, we reserve space for the left delimiter.

```

2648   \int_if_zero:nTF { \c@jCol }
2649   {
2650     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2651     {

```

In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.

```

2652       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2653       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2654       \@@_rec_preamble:n #2
2655     }
2656     {
2657       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2658       \@@_make_preamble_iv:nn { #1 } { #2 }
2659     }
2660   }
2661   { \@@_make_preamble_iv:nn { #1 } { #2 } }
2662 }
2663 \cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
2664 \cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
2665 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2666 {
2667   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2668   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2669   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2670   {
2671     \@@_error:nn { delimiter~after~opening } { #2 }
2672     \@@_rec_preamble:n

```

```

2673     }
2674     { \@@_rec_preamble:n #2 }
2675 }

```

In fact, it would be possible to define `\left` and `\right` as no-op.

```

2676 \cs_new_protected:cpn { @@ _ \token_to_str:N \left : } #1
2677 { \use:c { @@ _ \token_to_str:N ( : } }

```

For the closing delimiters. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have a opening delimiter following and we also have to see whether we are at the end of the preamble because, in that case, our letter must be considered as the right delimiter of the environment if the environment is `{NiceArray}`).

```

2678 \cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
2679 {
2680   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2681   \tl_if_in:nnTF { ) ] \} } { #2 }
2682   { \@@_make_preamble_v:nnn #1 #2 }
2683   {
2684     \str_if_eq:nnTF { \s_stop } { #2 }
2685     {
2686       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2687       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2688       {
2689         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2690         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2691         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2692         \@@_rec_preamble:n #2
2693       }
2694     }
2695     {
2696       \tl_if_in:nnT { ( [ \{ \left } } { #2 }
2697       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2698       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2699       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2700       \@@_rec_preamble:n #2
2701     }
2702   }
2703 }
2704 \cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
2705 \cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
2706 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2707 {
2708   \str_if_eq:nnTF { \s_stop } { #3 }
2709   {
2710     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2711     {
2712       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2713       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2714       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2715       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2716     }
2717     {
2718       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2719       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2720       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2721       \@@_error:nn { double~closing~delimiter } { #2 }
2722     }
2723   }
2724   {
2725     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2726     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2727     \@@_error:nn { double~closing~delimiter } { #2 }
2728     \@@_rec_preamble:n #3

```



```

2729     }
2730 }

2731 \cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
2732 { \use:c { @@ _ \token_to_str:N ) : } }

```

After a specifier of column, we have to test whether there is one or several <{...} because, after those potential <{...}, we have to insert !{\skip_horizontal:N ...} when the key `vlines` is used. In fact, we have also to test whether there is, after the <{...}, a @{...}.

```

2733 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2734 {
2735   \str_if_eq:nnTF { #1 } { < }
2736   { \@@_rec_preamble_after_col_i:n }
2737   {
2738     \str_if_eq:nnTF { #1 } { @ }
2739     { \@@_rec_preamble_after_col_ii:n }
2740     {
2741       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2742       {
2743         \tl_gput_right:Nn \g_@@_array_preamble_tl
2744         { ! { \skip_horizontal:N \arrayrulewidth } }
2745       }
2746       {
2747         \clist_if_in:NeT \l_@@_vlines_clist
2748         { \int_eval:n { \c@jCol + 1 } }
2749         {
2750           \tl_gput_right:Nn \g_@@_array_preamble_tl
2751           { ! { \skip_horizontal:N \arrayrulewidth } }
2752         }
2753       }
2754       \@@_rec_preamble:n { #1 }
2755     }
2756   }
2757 }

2758 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2759 {
2760   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2761   \@@_rec_preamble_after_col:n
2762 }

```

We have to catch a @{...} after a specifier of column because, if we have to draw a vertical rule, we have to add in that @{...} a `\hskip` corresponding to the width of the vertical rule.

```

2763 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2764 {
2765   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2766   {
2767     \tl_gput_right:Nn \g_@@_array_preamble_tl
2768     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2769   }
2770   {
2771     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2772     {
2773       \tl_gput_right:Nn \g_@@_array_preamble_tl
2774       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2775     }
2776     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2777   }
2778   \@@_rec_preamble:n
2779 }

2780 \cs_new_protected:cpn { @@ _ * : } #1 #2 #3

```

```

2781 {
2782   \tl_clear:N \l_tmpa_tl
2783   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2784   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2785 }

```

The token `\NC@find` is at the head of the definition of the columns type done by `\newcolumntype`. We want that token to be no-op here.

```

2786 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find : } #1
2787 { \@@_rec_preamble:n }

```

For the case of a letter `X`. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a `[` after the letter `X`.

```

2788 \cs_new_protected:Npn \@@_X: #1 #2
2789 {
2790   \str_if_eq:nnTF { #2 } { [ ]
2791     { \@@_make_preamble_X:w [ ] }
2792     { \@@_make_preamble_X:w [ ] #2 }
2793 }
2794 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2795 { \@@_make_preamble_X_i:n { #1 } }

```

`#1` is the optional argument of the `X` specifier (a list of *key-value* pairs).

The following set of keys is for the specifier `X` in the preamble of the array. Such specifier may have as keys all the keys of `{ nicematrix / p-column }` but also a key `V` and also a key which corresponds to a positive number (1, 2, 0.5, etc.) which is the *weight* of the columns. The following set of keys will be used to retrieve that value and store it in `\l_tmpa_fp`.

```

2796 \keys_define:nn { nicematrix / X-column }
2797 {
2798   V .code:n =
2799     \IfPackageLoadedTF { varwidth }
2800     {
2801       \bool_set_true:N \l_@@_V_of_X_bool
2802       \bool_gset_true:N \g_@@_V_of_X_bool
2803     }
2804     { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
2805   unknown .code:n =
2806     \regex_match:nVTF { \A[0-9]*\.[0-9]*\Z } \l_keys_key_str
2807     { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
2808     { \@@_error_or_warning:n { invalid~weight } }
2809 }

```

In the following command, `#1` is the list of the options of the specifier `X`.

```

2810 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2811 {

```

The possible values of `\l_@@_hpos_col_str` are `j` (for *justified* which is the initial value), `l`, `c` and `r` (when the user has used the corresponding key in the optional argument of the specifier `X`).

```

2812   \str_set:Nn \l_@@_hpos_col_str { j }

```

The possible values of `\l_@@_vpos_col_str` are `p` (the initial value), `m` and `b` (when the user has used the corresponding key in the optional argument of the specifier `X`).

```

2813   \str_set:Nn \l_@@_vpos_col_str { p }

```

We will store in `\l_tmpa_fp` the weight of the column (`\l_tmpa_fp` also appears in `{nicematrix/X-column}`) and the error message `invalid~weight`.

```

2814   \fp_set:Nn \l_tmpa_fp { 1.0 }
2815   \@@_keys_p_column:n { #1 }

```

The unknown keys have been stored by `\@@_keys_p_column:n` in `\l_tmpa_tl` and we use them right now in the set of keys `nicematrix/X-column` in order to retrieve the potential weight explicitly provided by the final user.

```
2816 \bool_set_false:N \l_@@_V_of_X_bool
2817 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

Now, the weight of the column is stored in `\l_tmpa_tl`.

```
2818 \fp_gadd:Nn \g_@@_total_X_weight_fp \l_tmpa_fp
```

We test whether we know the actual width of the X-columns by reading the `aux` file (after the first compilation, the width of the X-columns is computed and written in the `aux` file).

```
2819 \bool_if:NTF \l_@@_X_columns_aux_bool
2820 {
2821 \@@_make_preamble_ii_iv:nnn
```

Of course, the weight of a column depends of its weight (in `\l_tmpa_fp`).

```
2822 { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
2823 { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
2824 { \@@_no_update_width: }
2825 }
```

In the current compilation, we don't know the actual width of the X column. However, you have to construct the cells of that column! By convention, we have decided to compose in a `{minipage}` of width 5 cm even though we will nullify `\l_@@_cell_box` after its composition.

```
2826 {
2827 \tl_gput_right:Nn \g_@@_array_preamble_tl
2828 {
2829 > {
2830 \@@_cell_begin:
2831 \bool_set_true:N \l_@@_X_bool
```

You encounter a problem on 2023-03-04: for an environment with X columns, during the first compilations (which are not the definitive one), sometimes, some cells are declared empty even if they should not. That's a problem because user's instructions may use these nodes. That's why we have added the following `\NotEmpty`.

```
2832 \NotEmpty
```

The following code will nullify the box of the cell.

```
2833 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2834 { \hbox_set:Nn \l_@@_cell_box { } }
```

We put a `{minipage}` to give to the user the ability to put a command such as `\centering` in the `\RowStyle`.

```
2835 \begin { minipage } { 5 cm } \arraybackslash
2836 }
2837 c
2838 < {
2839 \end { minipage }
2840 \@@_cell_end:
2841 }
2842 }
2843 \int_gincr:N \c@jCol
2844 \@@_rec_preamble_after_col:n
2845 }
2846 }
```

```
2847 \cs_new_protected:Npn \@@_no_update_width:
2848 {
2849 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2850 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2851 }
```

For the letter set by the user with `vlines-in-sub-matrix` (`vlism`).

```

2852 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2853 {
2854   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2855   { \int_eval:n { \c@jCol + 1 } }
2856   \tl_gput_right:Ne \g_@@_array_preamble_tl
2857   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2858   \@@_rec_preamble:n
2859 }

```

The token `\s_stop` is a marker that we have inserted to mark the end of the preamble (as provided by the final user) that we have inserted in the TeX flow.

```

2860 \cs_set_eq:cn { @@ _ \token_to_str:N \s_stop : } \use_none:n

```

The following lines try to catch some errors (when the final user has forgotten the preamble of its environment).

```

2861 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
2862 { \@@_fatal:n { Preamble-forgotten } }
2863 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
2864 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
2865 { @@ _ \token_to_str:N \hline : }
2866 \cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
2867 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
2868 { @@ _ \token_to_str:N \hline : }
2869 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
2870 { @@ _ \token_to_str:N \hline : }
2871 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
2872 { @@ _ \token_to_str:N \hline : }
2873 \cs_set_eq:cc { @@ _ \token_to_str:N & : }
2874 { @@ _ \token_to_str:N \hline : }

```

12 The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```

2875 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2876 {

```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```

2877   \multispan { #1 }
2878   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2879   \begingroup
2880   \bool_if:NT \c_@@_testphase_table_bool
2881   { \tbl_update_multicolumn_cell_data:n { #1 } }
2882   \def \@@addamp
2883   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preammerr 5 } }

```

Now, we patch the (small) preamble as we have done with the main preamble of the array.

```

2884   \tl_gclear:N \g_@@_preamble_tl
2885   \@@_make_m_preamble:n #2 \q_stop

```

The following lines are an adaptation of the definition of `\multicolumn` in `array`.

```

2886   \exp_args:No \@mkpream \g_@@_preamble_tl
2887   \@addtopreamble \@empty
2888   \endgroup
2889   \bool_if:NT \c_@@_recent_array_bool
2890   { \UseTaggingSocket { tbl / colspan } { #1 } }

```

Now, we do a treatment specific to `nicematrix` which has no equivalent in the original definition of `\multicolumn`.

```

2891   \int_compare:nNnT { #1 } > { \c_one_int }
2892   {
2893     \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2894     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2895     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2896     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2897     {
2898       {
2899         \int_if_zero:nTF { \c@jCol }
2900         { \int_eval:n { \c@iRow + 1 } }
2901         { \int_use:N \c@iRow }
2902       }
2903       { \int_eval:n { \c@jCol + 1 } }
2904       {
2905         \int_if_zero:nTF { \c@jCol }
2906         { \int_eval:n { \c@iRow + 1 } }
2907         { \int_use:N \c@iRow }
2908       }
2909       { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2910     { }
2911   }
2912 }

```

We want `\cellcolor` to be available in `\multicolumn` because `\cellcolor` of `colortbl` is available in `\multicolumn`.

```

2913   \RenewDocumentCommand { \cellcolor } { 0 { } m }
2914   {
2915     \tl_gput_right:Ne \g_@@_pre_code_before_tl
2916     {
2917       \@@_rectanglecolor [ ##1 ]
2918       { \exp_not:n { ##2 } }
2919       { \int_use:N \c@iRow - \int_use:N \c@jCol }
2920       { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2921     }
2922     \ignorespaces
2923   }

```

The following lines were in the original definition of `\multicolumn`.

```

2924   \def \@sharp { #3 }
2925   \@arstrut
2926   \@preamble
2927   \null

```

We add some lines.

```

2928   \int_gadd:Nn \c@jCol { #1 - 1 }
2929   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
2930   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2931   \ignorespaces
2932 }

```

The following commands will patch the (small) preamble of the `\multicolumn`. All those commands have a `m` in their name to recall that they deal with the redefinition of `\multicolumn`.

```

2933   \cs_new_protected:Npn \@_make_m_preamble:n #1
2934   {
2935     \str_case:nnF { #1 }
2936     {
2937       c { \@_make_m_preamble_i:n #1 }
2938       l { \@_make_m_preamble_i:n #1 }

```

```

2939     r { \@@_make_m_preamble_i:n #1 }
2940     > { \@@_make_m_preamble_ii:nn #1 }
2941     ! { \@@_make_m_preamble_ii:nn #1 }
2942     @ { \@@_make_m_preamble_ii:nn #1 }
2943     | { \@@_make_m_preamble_iii:n #1 }
2944     p { \@@_make_m_preamble_iv:nnn t #1 }
2945     m { \@@_make_m_preamble_iv:nnn c #1 }
2946     b { \@@_make_m_preamble_iv:nnn b #1 }
2947     w { \@@_make_m_preamble_v:nnnn { } #1 }
2948     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2949     \q_stop { }
2950   }
2951   {
2952     \cs_if_exist:cTF { NC @ find @ #1 }
2953     {
2954       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2955       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2956     }
2957     {
2958       \str_if_eq:nnTF { #1 } { S }
2959       { \@@_fatal:n { unknown~column~type~S~multicolumn } }
2960       { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
2961     }
2962   }
2963 }

```

For c, l and r

```

2964 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2965   {
2966     \tl_gput_right:Nn \g_@@_preamble_tl
2967     {
2968       > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
2969       #1
2970       < \@@_cell_end:
2971     }

```

We test for the presence of a <.

```

2972     \@@_make_m_preamble_x:n
2973   }

```

For >, ! and @

```

2974 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2975   {
2976     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2977     \@@_make_m_preamble:n
2978   }

```

For |

```

2979 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2980   {
2981     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2982     \@@_make_m_preamble:n
2983   }

```

For p, m and b

```

2984 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2985   {
2986     \tl_gput_right:Nn \g_@@_preamble_tl
2987     {
2988       > {
2989         \@@_cell_begin:

```

We use `\setlength` instead of `\dim_set:N` in order to allow a specifier of column like `p{\widthof{Some words}}`. `widthof` is a command provided by `calc`. Of course, even if `calc` is not loaded, the following code will work with the standard version of `\setlength`.

```

2990         \setlength { \l_tmpa_dim } { #3 }
2991         \begin { minipage } [ #1 ] { \l_tmpa_dim }
2992         \mode_leave_vertical:
2993         \arraybackslash
2994         \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
2995     }
2996     c
2997     < {
2998         \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
2999         \end { minipage }
3000         \@_cell_end:
3001     }
3002 }

```

We test for the presence of a `<`.

```

3003     \@_make_m_preamble_x:n
3004 }

```

For `w` and `W`

```

3005 \cs_new_protected:Npn \@_make_m_preamble_v:nnnn #1 #2 #3 #4
3006 {
3007     \tl_gput_right:Nn \g_@@_preamble_tl
3008     {
3009         > {
3010             \dim_set:Nn \l_@@_col_width_dim { #4 }
3011             \hbox_set:Nw \l_@@_cell_box
3012             \@_cell_begin:
3013             \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
3014         }
3015         c
3016         < {
3017             \@_cell_end:
3018             \hbox_set_end:
3019             \bool_if:NT \g_@@_rotate_bool { \@_rotate_cell_box: }
3020             #1
3021             \@_adjust_size_box:
3022             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3023         }
3024     }

```

We test for the presence of a `<`.

```

3025     \@_make_m_preamble_x:n
3026 }

```

After a specifier of column, we have to test whether there is one or several `<{...}`.

```

3027 \cs_new_protected:Npn \@_make_m_preamble_x:n #1
3028 {
3029     \str_if_eq:nnTF { #1 } { < }
3030     { \@_make_m_preamble_ix:n }
3031     { \@_make_m_preamble:n { #1 } }
3032 }
3033 \cs_new_protected:Npn \@_make_m_preamble_ix:n #1
3034 {
3035     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3036     \@_make_m_preamble_x:n
3037 }

```

The command `\@@_put_box_in_flow:` puts the box `\l_tmpa_box` (which contains the array) in the flow. It is used for the environments with delimiters. First, we have to modify the height and the

depth to take back into account the potential exterior rows (the total height of the first row has been computed in `\l_tmpa_dim` and the total height of the potential last row in `\l_tmpb_dim`).

```

3038 \cs_new_protected:Npn \@@_put_box_in_flow:
3039 {
3040   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3041   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3042   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3043     { \box_use_drop:N \l_tmpa_box }
3044     { \@@_put_box_in_flow_i: }
3045 }

```

The command `\@@_put_box_in_flow_i:` is used when the value of `\l_@@_baseline_tl` is different of `c` (the initial value).

```

3046 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3047 {
3048   \pgfpicture
3049     \@@_qpoint:n { row - 1 }
3050     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3051     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3052     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3053     \dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim }

```

Now, `\g_tmpa_dim` contains the y -value of the center of the array (the delimiters are centered in relation with this value).

```

3054   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3055     {
3056       \int_set:Nn \l_tmpa_int
3057         {
3058           \str_range:Nnn
3059             \l_@@_baseline_tl
3060             6
3061             { \tl_count:o \l_@@_baseline_tl }
3062         }
3063       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3064     }
3065     {
3066       \str_if_eq:eeTF \l_@@_baseline_tl { t }
3067       { \int_set_eq:NN \l_tmpa_int \c_one_int }
3068       {
3069         \str_if_eq:onTF \l_@@_baseline_tl { b }
3070         { \int_set_eq:NN \l_tmpa_int \c@iRow }
3071         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3072       }
3073       \bool_lazy_or:nnT
3074       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3075       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3076       {
3077         \@@_error:n { bad-value~for~baseline }
3078         \int_set_eq:NN \l_tmpa_int \c_one_int
3079       }
3080       \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3081       \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3082     }
3083     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

Now, `\g_tmpa_dim` contains the value of the y translation we have to to.

```

3084   \endpgfpicture
3085   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3086   \box_use_drop:N \l_tmpa_box
3087 }

```


The following command is *always* used by `{NiceArrayWithDelims}` (even if, in fact, there is no tabular notes: in fact, it's not possible to know whether there is tabular notes or not before the composition of the blocks).

```
3088 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3089 {
```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```
3090 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3091 {
3092   \int_compare:nNnT { \c@jCol } > { \c_one_int }
3093   {
3094     \box_set_wd:Nn \l_@@_the_array_box
3095     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3096   }
3097 }
```

We need a `{minipage}` because we will insert a LaTeX list for the tabular notes (that means that a `\vtop{\hsize=...}` is not enough).

```
3098 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3099 \bool_if:NT \l_@@_caption_above_bool
3100 {
3101   \tl_if_empty:NF \l_@@_caption_tl
3102   {
3103     \bool_set_false:N \g_@@_caption_finished_bool
3104     \int_gzero:N \c@tabularnote
3105     \@@_insert_caption:
```

If there is one or several commands `\tabularnote` in the caption, we will write in the aux file the number of such tabular notes... but only the tabular notes for which the command `\tabularnote` has been used without its optional argument (between square brackets).

```
3106   \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
3107   {
3108     \tl_gput_right:Ne \g_@@_aux_tl
3109     {
3110       \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3111       { \int_use:N \g_@@_notes_caption_int }
3112     }
3113     \int_gzero:N \g_@@_notes_caption_int
3114   }
3115 }
3116 }
```

The `\hbox` avoids that the `pgfpicture` inside `\@@_draw_blocks` adds a extra vertical space before the notes.

```
3117 \hbox
3118 {
3119   \box_use_drop:N \l_@@_the_array_box
```

We have to draw the blocks right now because there may be tabular notes in some blocks (which are not mono-column: the blocks which are mono-column have been composed in boxes yet)... and we have to create (potentially) the extra nodes before creating the blocks since there are `medium` nodes to create for the blocks.

```
3120 \@@_create_extra_nodes:
3121 \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
3122 }
```

We don't do the following test with `\c@tabularnote` because the value of that counter is not reliable when the command `\ttabbox` of `floatrow` is used (because `\ttabbox` de-activate `\stepcounter` because it compiles twice its tabular).

```
3123 \bool_lazy_any:nT
3124 {
3125   { ! \seq_if_empty_p:N \g_@@_notes_seq }
```

```

3126     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3127     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3128   }
3129   \@@_insert_tabularnotes:
3130   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3131   \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
3132   \end { minipage }
3133 }

```

```

3134 \cs_new_protected:Npn \@@_insert_caption:
3135 {
3136   \tl_if_empty:NF \l_@@_caption_tl
3137   {
3138     \cs_if_exist:NTF \@cptype
3139     { \@@_insert_caption_i: }
3140     { \@@_error:n { caption~outside~float } }
3141   }
3142 }

```

```

3143 \cs_new_protected:Npn \@@_insert_caption_i:
3144 {
3145   \group_begin:

```

The flag `\l_@@_in_caption_bool` affects only the behavior of the command `\tabularnote` when used in the caption.

```

3146   \bool_set_true:N \l_@@_in_caption_bool

```

The package `floatrow` does a redefinition of `\@makecaption` which will extract the caption from the tabular. However, the old version of `\@makecaption` has been stored by `floatrow` in `\FR@makecaption`. That's why we restore the old version.

```

3147   \IfPackageLoadedT { floatrow }
3148   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3149   \tl_if_empty:NTF \l_@@_short_caption_tl
3150   { \caption }
3151   { \caption [ \l_@@_short_caption_tl ] }
3152   { \l_@@_caption_tl }

```

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That's why, when the same tabular note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of tabular notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

```

3153   \bool_if:NF \g_@@_caption_finished_bool
3154   {
3155     \bool_gset_true:N \g_@@_caption_finished_bool
3156     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3157     \int_gzero:N \c@tabularnote
3158   }
3159   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3160   \group_end:
3161 }

```

```

3162 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3163 {
3164   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3165   \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
3166 }

```

```

3167 \cs_new_protected:Npn \@@_insert_tabularnotes:
3168 {
3169   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3170   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3171   \skip_vertical:N 0.65ex

```

The TeX group is for potential specifications in the `\l_@@_notes_code_before_tl`.

```

3172   \group_begin:
3173   \l_@@_notes_code_before_tl
3174   \tl_if_empty:NF \g_@@_tabularnote_tl
3175   {
3176     \g_@@_tabularnote_tl \par
3177     \tl_gclear:N \g_@@_tabularnote_tl
3178   }

```

We compose the tabular notes with a list of `enumitem`. The `\strut` and the `\unskip` are designed to give the ability to put a `\bottomrule` at the end of the notes with a good vertical space.

```

3179   \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
3180   {
3181     \bool_if:NTF \l_@@_notes_para_bool
3182     {
3183       \begin { tabularnotes* }
3184         \seq_map_inline:Nn \g_@@_notes_seq
3185         { \@@_one_tabularnote:nn ##1 }
3186         \strut
3187       \end { tabularnotes* }

```

The following `\par` is mandatory for the event that the user has put `\footnotesize` (for example) in the `notes/code-before`.

```

3188     \par
3189   }
3190   {
3191     \tabularnotes
3192     \seq_map_inline:Nn \g_@@_notes_seq
3193     { \@@_one_tabularnote:nn ##1 }
3194     \strut
3195     \endtabularnotes
3196   }
3197 }
3198 \unskip
3199 \group_end:
3200 \bool_if:NT \l_@@_notes_bottomrule_bool
3201 {
3202   \IfPackageLoadedTF { booktabs }
3203   {

```

The two dimensions `\aboverulesep` et `\heavyrulewidth` are parameters defined by `booktabs`.

```

3204     \skip_vertical:N \aboverulesep

```

`\CT@arc@` is the specification of color defined by `colortbl` but you use it even if `colortbl` is not loaded.

```

3205     { \CT@arc@ \hrule height \heavyrulewidth }
3206   }
3207   { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3208 }
3209 \l_@@_notes_code_after_tl
3210 \seq_gclear:N \g_@@_notes_seq
3211 \seq_gclear:N \g_@@_notes_in_caption_seq
3212 \int_gzero:N \c@tabularnote
3213 }

```

The following command will format (after the main tabular) one tabularnote (with the command `\item`). `#1` is the label (when the command `\tabularnote` has been used with an optional argument between square brackets) and `#2` is the text of the note. The second argument is provided by curryfication.

```

3214 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3215 {
3216   \tl_if_novalue:nTF { #1 }
3217   { \item }
3218   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3219 }

```

The case of `baseline` equal to `b`. Remember that, when the key `b` is used, the `{array}` (of `array`) is constructed with the option `t` (and not `b`). Now, we do the translation to take into account the option `b`.

```

3220 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3221 {
3222   \pgfpicture
3223     \@@_qpoint:n { row - 1 }
3224     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3225     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3226     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3227   \endpgfpicture
3228   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3229   \int_if_zero:nT { \l_@@_first_row_int }
3230     {
3231       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3232       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3233     }
3234   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3235 }

```

Now, the general case.

```

3236 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3237 {

```

We convert a value of `t` to a value of 1.

```

3238   \str_if_eq:eeT \l_@@_baseline_tl { t }
3239   { \tl_set:Nn \l_@@_baseline_tl { 1 } }

```

Now, we convert the value of `\l_@@_baseline_tl` (which should represent an integer) to an integer stored in `\l_tmpa_int`.

```

3240   \pgfpicture
3241   \@@_qpoint:n { row - 1 }
3242   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3243   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3244   {
3245     \int_set:Nn \l_tmpa_int
3246       {
3247         \str_range:Nnn
3248           \l_@@_baseline_tl
3249           { 6 }
3250           { \tl_count:o \l_@@_baseline_tl }
3251       }
3252     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3253   }
3254   {
3255     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3256     \bool_lazy_or:nnT
3257       { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
3258       { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
3259     {
3260       \@@_error:n { bad-value-for-baseline }
3261       \int_set:Nn \l_tmpa_int 1
3262     }
3263     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3264   }
3265   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3266   \endpgfpicture
3267   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3268   \int_if_zero:nT { \l_@@_first_row_int }
3269   {
3270     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3271     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3272   }
3273   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3274 }
```

The command `\@@_put_box_in_flow_bis:` is used when the option `delimiters/max-width` is used because, in this case, we have to adjust the widths of the delimiters. The arguments `#1` and `#2` are the delimiters specified by the user.

```
3275 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3276 {
```

We will compute the real width of both delimiters used.

```
3277 \dim_zero_new:N \l_@@_real_left_delim_dim
3278 \dim_zero_new:N \l_@@_real_right_delim_dim
3279 \hbox_set:Nn \l_tmpb_box
3280 {
3281 \m@th % added 2024/11/21
3282 \c_math_toggle_token
3283 \left #1
3284 \vcenter
3285 {
3286 \vbox_to_ht:nn
3287 { \box_ht_plus_dp:N \l_tmpa_box }
3288 { }
3289 }
3290 \right .
3291 \c_math_toggle_token
3292 }
3293 \dim_set:Nn \l_@@_real_left_delim_dim
3294 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3295 \hbox_set:Nn \l_tmpb_box
3296 {
3297 \m@th % added 2024/11/21
3298 \c_math_toggle_token
3299 \left .
3300 \vbox_to_ht:nn
3301 { \box_ht_plus_dp:N \l_tmpa_box }
3302 { }
3303 \right #2
3304 \c_math_toggle_token
3305 }
3306 \dim_set:Nn \l_@@_real_right_delim_dim
3307 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

Now, we can put the box in the TeX flow with the horizontal adjustments on both sides.

```
3308 \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
3309 \@@_put_box_in_flow:
3310 \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
3311 }
```

The construction of the array in the environment `{NiceArrayWithDelims}` is, in fact, done by the environment `{@@-light-syntax}` or by the environment `{@@-normal-syntax}` (whether the option `light-syntax` is in force or not). When the key `light-syntax` is not used, the construction is a standard environment (and, thus, it's possible to use verbatim in the array).

```
3312 \NewDocumentEnvironment { @@-normal-syntax } { }
```

First, we test whether the environment is empty. If it is empty, we raise a fatal error (it's only a security). In order to detect whether it is empty, we test whether the next token is `\end` and, if it's the case, we test if this is the end of the environment (if it is not, a standard error will be raised by LaTeX for incorrect nested environments).

```
3313 {
3314 \peek_remove_spaces:n
3315 {
3316 \peek_meaning:NTF \end
3317 { \@@_analyze_end:Nn }
```

```

3318     {
3319     \@@_transform_preamble:
Here is the call to \array (we have a dedicated macro \@@_array:n because of compatibility with
the classes revtex4-1 and revtex4-2).
3320     \@@_array:o \g_@@_array_preamble_tl
3321     }
3322   }
3323 }
3324 {
3325 \@@_create_col_nodes:
3326 \endarray
3327 }

```

When the key `light-syntax` is in force, we use an environment which takes its whole body as an argument (with the specifier `b`).

```

3328 \NewDocumentEnvironment { @@-light-syntax } { b }
3329 {

```

First, we test whether the environment is empty. It's only a security. Of course, this test is more easy than the similar test for the "normal syntax" because we have the whole body of the environment in `#1`.

```

3330 \tl_if_empty:nT { #1 }
3331 { \@@_fatal:n { empty-environment } }
3332 \tl_if_in:nnT { #1 } { & }
3333 { \@@_fatal:n { ampersand-in~light-syntax } }
3334 \tl_if_in:nnT { #1 } { \ }
3335 { \@@_fatal:n { double-backslash-in~light-syntax } }

```

Now, you extract the `\CodeAfter` of the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be caught in the value of `\g_nicematrix_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_nicematrix_code_after_tl`.

```

3336 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

The command `\array` is hidden somewhere in `\@@_light_syntax_i:w`.

```

3337 }

```

Now, the second part of the environment. We must leave these lines in the second part (and not put them in the first part even though we caught the whole body of the environment with an argument of type `b`) in order to have the columns `S` of `siunitx` working fine.

```

3338 {
3339 \@@_create_col_nodes:
3340 \endarray
3341 }
3342 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
3343 {
3344 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

```

The body of the array, which is stored in the argument `#1`, is now split into items (and *not* tokens).

```

3345 \seq_clear_new:N \l_@@_rows_seq

```

We rescan the character of end of line in order to have the correct catcode.

```

3346 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3347 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3348 { \seq_set_split:Nee }
3349 { \seq_set_split:Non }
3350 \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3351 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3352 \tl_if_empty:NF \l_tmpa_tl
3353 { \seq_put_right:No \l_@@_rows_seq \l_tmpa_tl }

```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to compute that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```
3354   \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
3355   { \int_set:Nn \l_@@_last_row_int { \seq_count:N \l_@@_rows_seq } }
```

The new value of the body (that is to say after replacement of the separators of rows and columns by `\` and `&`) of the environment will be stored in `\l_@@_new_body_tl` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`).

```
3356   \tl_build_begin:N \l_@@_new_body_tl
3357   \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3358   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3359   \@@_line_with_light_syntax:o \l_tmpa_tl
```

Now, the other rows (with the same treatment, excepted that we have to insert `\` between the rows).

```
3360   \seq_map_inline:Nn \l_@@_rows_seq
3361   {
3362     \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3363     \@@_line_with_light_syntax:n { #1 }
3364   }
3365   \tl_build_end:N \l_@@_new_body_tl
3366   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
3367   {
3368     \int_set:Nn \l_@@_last_col_int
3369     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3370   }
```

Now, we can construct the preamble: if the user has used the key `last-col`, we have the correct number of columns even though the user has used `last-col` without value.

```
3371   \@@_transform_preamble:
```

The call to `\array` is in the following command (we have a dedicated macro `\@@_array:` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```
3372   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3373   }
3374   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3375   {
3376     \seq_clear_new:N \l_@@_cells_seq
3377     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3378     \int_set:Nn \l_@@_nb_cols_int
3379     {
3380       \int_max:nn
3381       { \l_@@_nb_cols_int }
3382       { \seq_count:N \l_@@_cells_seq }
3383     }
3384     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3385     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3386     \seq_map_inline:Nn \l_@@_cells_seq
3387     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3388   }
3389   \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security). When this command is used, `#1` is, in fact, always `\end`.

```
3390   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3391   {
3392     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3393     { \@@_fatal:n { empty-environment } }
```

We repeat in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```
3394   \end { #2 }
3395 }
```

The command `\@@_create_col_nodes:` will construct a special last row. That last row is a false row used to create the col nodes and to fix the width of the columns (when the array is constructed with an option which specifies the width of the columns such as `columns-width`).

```
3396 \cs_new:Npn \@@_create_col_nodes:
3397 {
3398   \crr
3399   \int_if_zero:nT { \l_@@_first_col_int }
3400   {
3401     \omit
3402     \hbox_overlap_left:n
3403     {
3404       \bool_if:NT \l_@@_code_before_bool
3405         { \pgfsys@markposition { \@@_env: - col - 0 } }
3406       \pgfpicture
3407       \pgfrememberpicturepositiononpagetrue
3408       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3409       \str_if_empty:NF \l_@@_name_str
3410         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3411       \endpgfpicture
3412       \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
3413     }
3414     &
3415   }
3416   \omit
```

The following instruction must be put after the instruction `\omit`.

```
3417   \bool_gset_true:N \g_@@_row_of_col_done_bool
```

First, we put a col node on the left of the first column (of course, we have to do that *after* the `\omit`).

```
3418   \int_if_zero:nTF { \l_@@_first_col_int }
3419   {
3420     \@@_mark_position:n { 1 }
3421     \pgfpicture
3422     \pgfrememberpicturepositiononpagetrue
3423     \pgfcoordinate { \@@_env: - col - 1 }
3424       { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3425     \str_if_empty:NF \l_@@_name_str
3426       { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3427     \endpgfpicture
3428   }
3429   {
3430     \bool_if:NT \l_@@_code_before_bool
3431     {
3432       \hbox
3433       {
3434         \skip_horizontal:n { 0.5 \arrayrulewidth }
3435         \pgfsys@markposition { \@@_env: - col - 1 }
3436         \skip_horizontal:n { -0.5 \arrayrulewidth }
3437       }
3438     }
3439     \pgfpicture
3440     \pgfrememberpicturepositiononpagetrue
3441     \pgfcoordinate { \@@_env: - col - 1 }
3442       { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3443     \@@_node_alias:n { 1 }
3444     \endpgfpicture
3445   }
```


We compute in `\g_tmpa_skip` the common width of the columns (it's a skip and not a dimension). We use a global variable because we are in a cell of an `\halign` and because we have to use that variable in other cells (of the same row). The affectation of `\g_tmpa_skip`, like all the affectations, must be done after the `\omit` of the cell.

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but we will add some dimensions to it.

```

3446   \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3447   \bool_if:NF \l_@@_auto_columns_width_bool
3448     { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
3449     {
3450       \bool_lazy_and:nnTF
3451         { \l_@@_auto_columns_width_bool }
3452         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3453         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3454         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3455       \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3456     }
3457   \skip_horizontal:N \g_tmpa_skip
3458   \hbox
3459     {
3460       \@@_mark_position:n { 2 }
3461       \pgfpicture
3462       \pgfrememberpicturepositiononpagetrue
3463       \pgfcoordinate { \@@_env: - col - 2 }
3464         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3465       \@@_node_alias:n { 2 }
3466       \endpgfpicture
3467     }

```

We begin a loop over the columns. The integer `\g_tmpa_int` will be the number of the current column. This integer is used for the Tikz nodes.

```

3468   \int_gset_eq:NN \g_tmpa_int \c_one_int
3469   \bool_if:NTF \g_@@_last_col_found_bool
3470     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
3471     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
3472     {
3473       &
3474       \omit
3475       \int_gincr:N \g_tmpa_int

```

The incrementation of the counter `\g_tmpa_int` must be done after the `\omit` of the cell.

```

3476       \skip_horizontal:N \g_tmpa_skip
3477       \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }

```

We create the `col` node on the right of the current column.

```

3478       \pgfpicture
3479       \pgfrememberpicturepositiononpagetrue
3480       \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3481         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3482       \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3483       \endpgfpicture
3484     }

3485     &
3486     \omit

```

If there is only one column (and a potential “last column”), we don't have to put the following code (there is only one column and we have put the correct code previously).

```

3487   \bool_lazy_or:nnF
3488     { \int_compare_p:nNn \g_@@_col_total_int = 1 }
3489     { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
3490     {
3491       \skip_horizontal:N \g_tmpa_skip

```

```

3492 \int_gincr:N \g_tmpa_int
3493 \bool_lazy_any:nF
3494 {
3495   \g_@@_delims_bool
3496   \l_@@_tabular_bool
3497   { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3498   \l_@@_exterior_arraycolsep_bool
3499   \l_@@_bar_at_end_of_pream_bool
3500 }
3501 { \skip_horizontal:n { - \col@sep } }
3502 \bool_if:NT \l_@@_code_before_bool
3503 {
3504   \hbox
3505   {
3506     \skip_horizontal:n { -0.5 \arrayrulewidth }

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

3507   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3508   { \skip_horizontal:n { - \arraycolsep } }
3509   \pgfsys@markposition
3510   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3511   \skip_horizontal:n { 0.5 \arrayrulewidth }
3512   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3513   { \skip_horizontal:N \arraycolsep }
3514 }
3515 }
3516 \pgfpicture
3517 \pgfrememberpicturepositiononpagetrue
3518 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3519 {
3520   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3521   {
3522     \pgfpoint
3523     { - 0.5 \arrayrulewidth - \arraycolsep }
3524     \c_zero_dim
3525   }
3526   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3527 }
3528 \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
3529 \endpgfpicture
3530 }

```

```

3531 \bool_if:NT \g_@@_last_col_found_bool
3532 {
3533   \hbox_overlap_right:n
3534   {
3535     \skip_horizontal:N \g_@@_width_last_col_dim
3536     \skip_horizontal:N \col@sep
3537     \bool_if:NT \l_@@_code_before_bool
3538     {
3539       \pgfsys@markposition
3540       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3541     }
3542     \pgfpicture
3543     \pgfrememberpicturepositiononpagetrue
3544     \pgfcoordinate
3545     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3546     \pgfpointorigin
3547     \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
3548     \endpgfpicture
3549   }

```

```

3550     }
3551     % \cr
3552     }

3553 \cs_new_protected:Npn \@@_mark_position:n #1
3554 {
3555     \bool_if:NT \l_@@_code_before_bool
3556     {
3557         \hbox
3558         {
3559             \skip_horizontal:n { -0.5 \arrayrulewidth }
3560             \pgfsys@markposition { \@@_env: - col - #1 }
3561             \skip_horizontal:n { 0.5 \arrayrulewidth }
3562         }
3563     }
3564 }

3565 \cs_new_protected:Npn \@@_node_alias:n #1
3566 {
3567     \str_if_empty:NF \l_@@_name_str
3568     { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
3569 }

```

Here is the preamble for the “first column” (if the user uses the key `first-col`)

```

3570 \tl_const:Nn \c_@@_preamble_first_col_tl
3571 {
3572     >
3573     {

```

At the beginning of the cell, we link `\CodeAfter` to a command which begins with `\` (whereas the standard version of `\CodeAfter` begins does not).

```

3574         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3575         \bool_gset_true:N \g_@@_after_col_zero_bool
3576         \@@_begin_of_row:
3577         \hbox_set:Nw \l_@@_cell_box
3578         \@@_math_toggle:
3579         \@@_tuning_key_small:

```

We insert `\l_@@_code_for_first_col_tl...` but we don’t insert it in the potential “first row” and in the potential “last row”.

```

3580         \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3581         {
3582             \bool_lazy_or:nnT
3583             { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3584             { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3585             {
3586                 \l_@@_code_for_first_col_tl
3587                 \xglobal \colorlet { nicematrix-first-col } { . }
3588             }
3589         }
3590     }

```

Be careful: despite this letter `l` the cells of the “first column” are composed in a `R` manner since they are composed in a `\hbox_overlap_left:n`.

```

3591     l
3592     <
3593     {
3594         \@@_math_toggle:
3595         \hbox_set_end:
3596         \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3597         \@@_adjust_size_box:
3598         \@@_update_for_first_and_last_row:

```

We actualise the width of the “first column” because we will use this width after the construction of the array.

```

3599   \dim_gset:Nn \g_@@_width_first_col_dim
3600   { \dim_max:nn { \g_@@_width_first_col_dim } { \box_wd:N \l_@@_cell_box } }

```

The content of the cell is inserted in an overlapping position.

```

3601   \hbox_overlap_left:n
3602   {
3603     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3604     { \@@_node_cell: }
3605     { \box_use_drop:N \l_@@_cell_box }
3606     \skip_horizontal:N \l_@@_left_delim_dim
3607     \skip_horizontal:N \l_@@_left_margin_dim
3608     \skip_horizontal:N \l_@@_extra_left_margin_dim
3609   }
3610   \bool_gset_false:N \g_@@_empty_cell_bool
3611   \skip_horizontal:n { -2 \col@sep }
3612 }
3613 }

```

Here is the preamble for the “last column” (if the user uses the key `last-col`).

```

3614 \tl_const:Nn \c_@@_preamble_last_col_tl
3615 {
3616   >
3617   {
3618     \bool_set_true:N \l_@@_in_last_col_bool

```

At the beginning of the cell, we link `\CodeAfter` to a command which begins with `\` (whereas the standard version of `\CodeAfter` begins does not).

```

3619     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```

3620     \bool_gset_true:N \g_@@_last_col_found_bool
3621     \int_gincr:N \c@jCol
3622     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3623     \hbox_set:Nw \l_@@_cell_box
3624     \@@_math_toggle:
3625     \@@_tuning_key_small:

```

We insert `\l_@@_code_for_last_col_tl...` but we don’t insert it in the potential “first row” and in the potential “last row”.

```

3626     \int_compare:nNnT { \c@iRow } > { \c_zero_int }
3627     {
3628       \bool_lazy_or:nnT
3629       { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
3630       { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
3631       {
3632         \l_@@_code_for_last_col_tl
3633         \xglobal \colorlet { nicematrix-last-col } { . }
3634       }
3635     }
3636   }
3637   l
3638   <
3639   {
3640     \@@_math_toggle:
3641     \hbox_set_end:
3642     \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
3643     \@@_adjust_size_box:
3644     \@@_update_for_first_and_last_row:

```

We actualise the width of the “last column” because we will use this width after the construction of the array.

```

3645     \dim_gset:Nn \g_@@_width_last_col_dim
3646     { \dim_max:nn { \g_@@_width_last_col_dim } { \box_wd:N \l_@@_cell_box } }
3647     \skip_horizontal:n { -2 \col@sep }

```

The content of the cell is inserted in an overlapping position.

```

3648     \hbox_overlap_right:n
3649     {
3650         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
3651         {
3652             \skip_horizontal:N \l_@@_right_delim_dim
3653             \skip_horizontal:N \l_@@_right_margin_dim
3654             \skip_horizontal:N \l_@@_extra_right_margin_dim
3655             \@@_node_cell:
3656         }
3657     }
3658     \bool_gset_false:N \g_@@_empty_cell_bool
3659 }
3660 }

```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}`.

```

3661 \NewDocumentEnvironment { NiceArray } { }
3662 {
3663     \bool_gset_false:N \g_@@_delims_bool
3664     \str_if_empty:NT \g_@@_name_env_str
3665     { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

We put `.` and `.` for the delimiters but, in fact, that doesn't matter because these arguments won't be used in `{NiceArrayWithDelims}` (because the flag `\g_@@_delims_bool` is set to false).

```

3666     \NiceArrayWithDelims . .
3667 }
3668 { \endNiceArrayWithDelims }

```

We create the variants of the environment `{NiceArrayWithDelims}`.

```

3669 \cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
3670 {
3671     \NewDocumentEnvironment { #1 NiceArray } { }
3672     {
3673         \bool_gset_true:N \g_@@_delims_bool
3674         \str_if_empty:NT \g_@@_name_env_str
3675         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3676         \@@_test_if_math_mode:
3677         \NiceArrayWithDelims #2 #3
3678     }
3679     { \endNiceArrayWithDelims }
3680 }
3681 \@@_def_env:NNN p ( )
3682 \@@_def_env:NNN b [ ]
3683 \@@_def_env:NNN B \{ \}
3684 \@@_def_env:NNN v \vert \vert
3685 \@@_def_env:NNN V \Vert \Vert

```

13 The environment `{NiceMatrix}` and its variants

```

3686 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3687 {
3688     \bool_set_false:N \l_@@_preamble_bool
3689     \tl_clear:N \l_tmpa_tl
3690     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3691     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3692     \tl_put_right:Nn \l_tmpa_tl

```

```

3693 {
3694 *
3695 {
3696 \int_case:nnF \l_@@_last_col_int
3697 {
3698 { -2 } { \c@MaxMatrixCols }
3699 { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3700 }
3701 { \int_eval:n { \l_@@_last_col_int - 1 } }
3702 }
3703 { #2 }
3704 }
3705 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3706 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3707 }
3708 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3709 \clist_map_inline:nn { p , b , B , v , V }
3710 {
3711 \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3712 {
3713 \bool_gset_true:N \g_@@_delims_bool
3714 \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3715 \int_if_zero:nT { \l_@@_last_col_int }
3716 {
3717 \bool_set_true:N \l_@@_last_col_without_value_bool
3718 \int_set:Nn \l_@@_last_col_int { -1 }
3719 }
3720 \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3721 \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
3722 }
3723 { \use:c { end #1 NiceArray } }
3724 }

```

We define also an environment {NiceMatrix}

```

3725 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3726 {
3727 \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3728 \int_if_zero:nT { \l_@@_last_col_int }
3729 {
3730 \bool_set_true:N \l_@@_last_col_without_value_bool
3731 \int_set:Nn \l_@@_last_col_int { -1 }
3732 }
3733 \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3734 \bool_lazy_or:nnT
3735 { \clist_if_empty_p:N \l_@@_vlines_clist }
3736 { \l_@@_except_borders_bool }
3737 { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3738 \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
3739 }
3740 { \endNiceArray }

```

The following command will be linked to \NotEmpty in the environments of nicematrix.

```

3741 \cs_new_protected:Npn \@@_NotEmpty:
3742 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

14 {NiceTabular}, {NiceTabularX} and {NiceTabular*}

```

3743 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3744 {

```

If the dimension `\l_@@_width_dim` is equal to 0 pt, that means that it has not been set by a previous use of `\NiceMatrixOptions`.

```

3745 \dim_compare:nNt { \l_@@_width_dim } = { \c_zero_dim }
3746   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3747 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3748 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3749 \tl_if_empty:NF \l_@@_short_caption_tl
3750   {
3751     \tl_if_empty:NT \l_@@_caption_tl
3752     {
3753       \@@_error_or_warning:n { short-caption-without~caption }
3754       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3755     }
3756   }
3757 \tl_if_empty:NF \l_@@_label_tl
3758   {
3759     \tl_if_empty:NT \l_@@_caption_tl
3760     { \@@_error_or_warning:n { label~without~caption } }
3761   }
3762 \NewDocumentEnvironment { TabularNote } { b }
3763   {
3764     \bool_if:NTF \l_@@_in_code_after_bool
3765     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3766     {
3767       \tl_if_empty:NF \g_@@_tabularnote_tl
3768       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3769       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3770     }
3771   }
3772   { }
3773 \@@_settings_for_tabular:
3774 \NiceArray { #2 }
3775 }
3776 { \endNiceArray }
3777 \cs_new_protected:Npn \@@_settings_for_tabular:
3778   {
3779     \bool_set_true:N \l_@@_tabular_bool
3780     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3781     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3782     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3783   }

3784 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3785   {
3786     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3787     \dim_set:Nn \l_@@_width_dim { #1 }
3788     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3789     \@@_settings_for_tabular:
3790     \NiceArray { #3 }
3791   }
3792   {
3793     \endNiceArray
3794     \fp_compare:nNt { \g_@@_total_X_weight_fp } = { \c_zero_fp }
3795     { \@@_error:n { NiceTabularX~without~X } }
3796   }

3797 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3798   {
3799     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3800     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3801     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3802     \@@_settings_for_tabular:

```

```

3803   \NiceArray { #3 }
3804   }
3805   { \endNiceArray }

```

15 After the construction of the array

The following command will be used when the key `rounded-corners` is in force (this is the key `rounded-corners` for the whole environment and *not* the key `rounded-corners` of a command `\Block`).

```

3806 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3807   {
3808     \bool_lazy_all:nT
3809     {
3810       { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
3811       { \l_@@_hvlines_bool }
3812       { ! \g_@@_delims_bool }
3813       { ! \l_@@_except_borders_bool }
3814     }
3815     {
3816       \bool_set_true:N \l_@@_except_borders_bool
3817       \clist_if_empty:NF \l_@@_corners_clist
3818       { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3819       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3820       {
3821         \@@_stroke_block:nnn
3822         {
3823           rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3824           draw = \l_@@_rules_color_tl
3825         }
3826         { 1-1 }
3827         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3828       }
3829     }
3830   }

```

```

3831 \cs_new_protected:Npn \@@_after_array:
3832   {

```

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_ii:` in order to come back to the standard definition of `\multicolumn` (in the tabulars used by the final user in the cells of our array of `nicematrix`) and maybe another linked to `colortbl`.

```

3833   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3834   \group_begin:

```

When the option `last-col` is used in the environments with explicit preambles (like `{NiceArray}`, `{pNiceArray}`, etc.) a special type of column is used at the end of the preamble in order to compose the cells in an overlapping position (with `\hbox_overlap_right:n`) but (if `last-col` has been used), we don't have the number of that last column. However, we have to know that number for the color of the potential `\Vdots` drawn in that last column. That's why we fix the correct value of `\l_@@_last_col_int` in that case.

```

3835   \bool_if:NT \g_@@_last_col_found_bool
3836     { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```

If we are in an environment without preamble (like `{NiceMatrix}` or `{pNiceMatrix}`) and if the option `last-col` has been used without value we also fix the real value of `\l_@@_last_col_int`.

```

3837   \bool_if:NT \l_@@_last_col_without_value_bool
3838     { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }

```


It's also time to give to `\l_@@_last_row_int` its real value.

```

3839   \bool_if:NT \l_@@_last_row_without_value_bool
3840     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3841   \tl_gput_right:Ne \g_@@_aux_tl
3842     {
3843     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3844       {
3845         \int_use:N \l_@@_first_row_int ,
3846         \int_use:N \c@iRow ,
3847         \int_use:N \g_@@_row_total_int ,
3848         \int_use:N \l_@@_first_col_int ,
3849         \int_use:N \c@jCol ,
3850         \int_use:N \g_@@_col_total_int
3851       }
3852     }

```

We write also the potential content of `\g_@@_pos_of_blocks_seq`. It will be used to recreate the blocks with a name in the `\CodeBefore` and also if the command `\rowcolors` is used with the key `respect=blocks`).

```

3853   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3854     {
3855     \tl_gput_right:Ne \g_@@_aux_tl
3856       {
3857       \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3858         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3859       }
3860     }
3861   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3862     {
3863     \tl_gput_right:Ne \g_@@_aux_tl
3864       {
3865       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3866         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3867       \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3868         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3869       }
3870     }

```

Now, you create the diagonal nodes by using the row nodes and the col nodes.

```

3871   \@@_create_diag_nodes:

```

We create the aliases using `last` for the nodes of the cells in the last row and the last column.

```

3872   \pgfpicture
3873   \@@_create_aliases_last:
3874   \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
3875   \endpgfpicture

```

By default, the diagonal lines will be parallelized¹². There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

3876   \bool_if:NT \l_@@_parallelize_diags_bool
3877     {
3878     \int_gzero:N \g_@@_ddots_int
3879     \int_gzero:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

¹²It's possible to use the option `parallelize-diags` to disable this parallelization.

```

3880     \dim_gzero:N \g_@@_delta_x_one_dim
3881     \dim_gzero:N \g_@@_delta_y_one_dim
3882     \dim_gzero:N \g_@@_delta_x_two_dim
3883     \dim_gzero:N \g_@@_delta_y_two_dim
3884   }

3885   \bool_set_false:N \l_@@_initial_open_bool
3886   \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_xdots_radius_dim` and `\l_@@_xdots_inter_dim` (used to draw the dotted lines created by `\hdottedline` and `\vdottedline` and also for all the other dotted lines when `line-style` is equal to `standard`, which is the initial value) are changed.

```

3887   \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }

```

Now, we actually draw the dotted lines (specified by `\Cdots`, `\Vdots`, etc.).

```

3888   \@@_draw_dotted_lines:

```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_cells_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3889   \clist_if_empty:NF \l_@@_corners_clist
3890   {
3891     \bool_if:NTF \l_@@_no_cell_nodes_bool
3892     { \@@_error:n { corners~with~no~cell~nodes } }
3893     { \@@_compute_corners: }
3894   }

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3895   \@@_adjust_pos_of_blocks_seq:

3896   \@@_deal_with_rounded_corners:
3897   \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
3898   \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }

```

Now, the pre-code-after and then, the `\CodeAfter`.

```

3899   \IfPackageLoadedT { tikz }
3900   {
3901     \tikzset
3902     {
3903       every~picture / .style =
3904       {
3905         overlay ,
3906         remember~picture ,
3907         name~prefix = \@@_env: -
3908       }
3909     }
3910   }
3911   \bool_if:NT \c_@@_recent_array_bool
3912   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3913   \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3914   \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3915   \cs_set_eq:NN \OverBrace \@@_OverBrace
3916   \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3917   \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3918   \cs_set_eq:NN \line \@@_line

```

The LaTeX-style boolean `\ifmeasuring@` is used by `amsmath` during the phase of measure in environments such as `{align}`, etc.

```

3919   \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3920   \tl_gclear:N \g_@@_pre_code_after_tl

```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it's possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That's why we set `\CodeAfter` to be *no-op* now.

```
3921 \cs_set_eq:NN \CodeAfter \prg_do_nothing:
```

We clear the list of the names of the potential `\SubMatrix` that will appear in the `\CodeAfter` (unfortunately, that list has to be global).

```
3922 \seq_gclear:N \g_@@_submatrix_names_seq
```

The following code is a security for the case the user has used `babel` with the option `spanish`: in that case, the characters `>` and `<` are activated and Tikz is not able to solve the problem (even with the Tikz library `babel`).

```
3923 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3924 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }
```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

```
3925 \bool_set_true:N \l_@@_in_code_after_bool
3926 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3927 \scan_stop:
3928 \tl_gclear:N \g_nicematrix_code_after_tl
3929 \group_end:
```

`\g_@@_pre_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor`. These instructions will be written on the aux file to be added to the code-before in the next run.

```
3930 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3931 \tl_if_empty:NF \g_@@_pre_code_before_tl
3932 {
3933   \tl_gput_right:Ne \g_@@_aux_tl
3934   {
3935     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3936     { \exp_not:o \g_@@_pre_code_before_tl }
3937   }
3938   \tl_gclear:N \g_@@_pre_code_before_tl
3939 }
3940 \tl_if_empty:NF \g_nicematrix_code_before_tl
3941 {
3942   \tl_gput_right:Ne \g_@@_aux_tl
3943   {
3944     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3945     { \exp_not:o \g_nicematrix_code_before_tl }
3946   }
3947   \tl_gclear:N \g_nicematrix_code_before_tl
3948 }

3949 \str_gclear:N \g_@@_name_env_str
3950 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹³. This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That's why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```
3951 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3952 }
```

¹³e.g. `\color[rgb]{0.5,0.5,0}`

```

3953 \cs_new_protected:Npn \@_tuning_key_small_for_dots:
3954 {
3955   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3956   \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_end_dim` correspond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```

3957   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3958     { 0.6 \l_@@_xdots_shorten_start_dim }
3959   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3960     { 0.6 \l_@@_xdots_shorten_end_dim }
3961 }

```

The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```

3962 \NewDocumentCommand \@_CodeAfter_keys: { 0 { } }
3963 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

```

3964 \cs_new_protected:Npn \@_create_alias_nodes:
3965 {
3966   \int_step_inline:nn { \c@iRow }
3967   {
3968     \pgfnodealias
3969       { \l_@@_name_str - ##1 - last }
3970       { \@_env: - ##1 - \int_use:N \c@jCol }
3971   }
3972   \int_step_inline:nn { \c@jCol }
3973   {
3974     \pgfnodealias
3975       { \l_@@_name_str - last - ##1 }
3976       { \@_env: - \int_use:N \c@iRow - ##1 }
3977   }
3978   \pgfnodealias % added 2025-04-05
3979     { \l_@@_name_str - last - last }
3980     { \@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
3981 }

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It’s possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```

3982 \cs_new_protected:Npn \@_adjust_pos_of_blocks_seq:
3983 {
3984   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3985     { \@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3986 }

```

The following command must *not* be protected.

```

3987 \cs_new:Npn \@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3988 {
3989   { #1 }
3990   { #2 }
3991   {
3992     \int_compare:nNnTF { #3 } > { 98 }
3993       { \int_use:N \c@iRow }
3994       { #3 }
3995   }

```

```

3996 {
3997   \int_compare:nNnTF { #4 } > { 98 }
3998     { \int_use:N \c@jCol }
3999     { #4 }
4000 }
4001 { #5 }
4002 }

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That’s why we have to define the adequate version of `\@@_draw_dotted_lines`: whether Tikz is loaded or not (in that case, only PGF is loaded).

```

4003 \hook_gput_code:nnn { begindocument } { . }
4004 {
4005   \cs_new_protected:Npe \@@_draw_dotted_lines:
4006     {
4007       \c_@@_pgfortikzpicture_tl
4008       \@@_draw_dotted_lines_i:
4009       \c_@@_endpgfortikzpicture_tl
4010     }
4011 }

```

The following command *must* be protected because it will appear in the construction of the command `\@@_draw_dotted_lines:`.

```

4012 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4013 {
4014   \pgfrememberpicturepositiononpagetrue
4015   \pgf@relevantforpicturesizefalse
4016   \g_@@_HVdotsfor_lines_tl
4017   \g_@@_Vdots_lines_tl
4018   \g_@@_Ddots_lines_tl
4019   \g_@@_Iddots_lines_tl
4020   \g_@@_Cdots_lines_tl
4021   \g_@@_Ldots_lines_tl
4022 }

4023 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4024 {
4025   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4026   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4027 }

```

We define a new PGF shape for the diag nodes because we want to provide an anchor called `.5` for those nodes.

```

4028 \pgfdeclareshape { @@_diag_node }
4029 {
4030   \savedanchor { \five }
4031   {
4032     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4033     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4034   }
4035   \anchor { 5 } { \five }
4036   \anchor { center } { \pgfpointorigin }
4037   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4038   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4039   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4040   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4041   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4042   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4043   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4044   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4045   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4046   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4047 }

```


The command `\l_@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
4089 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4090 {
```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

```
4091 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4092 \int_set:Nn \l_@@_initial_i_int { #1 }
4093 \int_set:Nn \l_@@_initial_j_int { #2 }
4094 \int_set:Nn \l_@@_final_i_int { #1 }
4095 \int_set:Nn \l_@@_final_j_int { #2 }
```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```
4096 \bool_set_false:N \l_@@_stop_loop_bool
4097 \bool_do_until:Nn \l_@@_stop_loop_bool
4098 {
4099   \int_add:Nn \l_@@_final_i_int { #3 }
4100   \int_add:Nn \l_@@_final_j_int { #4 }
4101   \bool_set_false:N \l_@@_final_open_bool
```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```
4102   \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4103     \if_int_compare:w #3 = \c_one_int
4104       \bool_set_true:N \l_@@_final_open_bool
4105     \else:
4106       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4107         \bool_set_true:N \l_@@_final_open_bool
4108       \fi:
4109     \fi:
4110   \else:
4111     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4112       \if_int_compare:w #4 = -1
4113         \bool_set_true:N \l_@@_final_open_bool
4114       \fi:
4115     \else:
4116       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4117         \if_int_compare:w #4 = \c_one_int
4118           \bool_set_true:N \l_@@_final_open_bool
4119         \fi:
4120       \fi:
4121     \fi:
4122   \fi:
```

```
4123     \bool_if:NTF \l_@@_final_open_bool
```

If we are outside the matrix, we have found the extremity of the dotted line and it's an *open* extremity.

```
4124     {
```

We do a step backwards.

```
4125         \int_sub:Nn \l_@@_final_i_int { #3 }
4126         \int_sub:Nn \l_@@_final_j_int { #4 }
4127         \bool_set_true:N \l_@@_stop_loop_bool
4128     }
```

If we are in the matrix, we test whether the cell is empty. If it's not the case, we stop the loop because we have found the correct values for `\l_@@_final_i_int` and `\l_@@_final_j_int`.

```
4129     {
4130         \cs_if_exist:cTF
4131         {
4132             @@ _ dotted _
4133             \int_use:N \l_@@_final_i_int -
4134             \int_use:N \l_@@_final_j_int
4135         }
4136         {
4137             \int_sub:Nn \l_@@_final_i_int { #3 }
4138             \int_sub:Nn \l_@@_final_j_int { #4 }
4139             \bool_set_true:N \l_@@_final_open_bool
4140             \bool_set_true:N \l_@@_stop_loop_bool
4141         }
4142     }
4143     \cs_if_exist:cTF
4144     {
4145         pgf @ sh @ ns @ \@@_env:
4146         - \int_use:N \l_@@_final_i_int
4147         - \int_use:N \l_@@_final_j_int
4148     }
4149     { \bool_set_true:N \l_@@_stop_loop_bool }
```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don't want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```
4150     {
4151         \cs_set_nopar:cpn
4152         {
4153             @@ _ dotted _
4154             \int_use:N \l_@@_final_i_int -
4155             \int_use:N \l_@@_final_j_int
4156         }
4157     }
4158 }
4159 }
4160 }
4161 }
```

For `\l_@@_initial_i_int` and `\l_@@_initial_j_int` the programming is similar to the previous one.

```
4162     \bool_set_false:N \l_@@_stop_loop_bool
```

The following line of code is only for efficiency in the following loop.

```
4163     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4164     \bool_do_until:Nn \l_@@_stop_loop_bool
4165     {
4166         \int_sub:Nn \l_@@_initial_i_int { #3 }
4167         \int_sub:Nn \l_@@_initial_j_int { #4 }
4168         \bool_set_false:N \l_@@_initial_open_bool
```


We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4169     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4170     \if_int_compare:w #3 = \c_one_int
4171     \bool_set_true:N \l_@@_initial_open_bool
4172     \else:
\l_tmpa_int contains \l_@@_col_min_int - 1 (only for efficiency).
4173     \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4174     \bool_set_true:N \l_@@_initial_open_bool
4175     \fi:
4176     \fi:
4177     \else:
4178     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4179     \if_int_compare:w #4 = \c_one_int
4180     \bool_set_true:N \l_@@_initial_open_bool
4181     \fi:
4182     \else:
4183     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4184     \if_int_compare:w #4 = -1
4185     \bool_set_true:N \l_@@_initial_open_bool
4186     \fi:
4187     \fi:
4188     \fi:
4189     \fi:
4190     \bool_if:NTF \l_@@_initial_open_bool
4191     {
4192     \int_add:Nn \l_@@_initial_i_int { #3 }
4193     \int_add:Nn \l_@@_initial_j_int { #4 }
4194     \bool_set_true:N \l_@@_stop_loop_bool
4195     }
4196     {
4197     \cs_if_exist:cTF
4198     {
4199     @@ _ dotted _
4200     \int_use:N \l_@@_initial_i_int -
4201     \int_use:N \l_@@_initial_j_int
4202     }
4203     {
4204     \int_add:Nn \l_@@_initial_i_int { #3 }
4205     \int_add:Nn \l_@@_initial_j_int { #4 }
4206     \bool_set_true:N \l_@@_initial_open_bool
4207     \bool_set_true:N \l_@@_stop_loop_bool
4208     }
4209     {
4210     \cs_if_exist:cTF
4211     {
4212     pgf @ sh @ ns @ \@@_env:
4213     - \int_use:N \l_@@_initial_i_int
4214     - \int_use:N \l_@@_initial_j_int
4215     }
4216     { \bool_set_true:N \l_@@_stop_loop_bool }
4217     {
4218     \cs_set_nopar:cpn
4219     {
4220     @@ _ dotted _
4221     \int_use:N \l_@@_initial_i_int -
4222     \int_use:N \l_@@_initial_j_int
4223     }
4224     { }
4225     }
4226     }
4227     }

```

```
4228     }
```

We remind the rectangle described by all the dotted lines in order to respect the corresponding virtual “block” when drawing the horizontal and vertical rules.

```
4229     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4230     {
4231     { \int_use:N \l_@@_initial_i_int }
```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```
4232     { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4233     { \int_use:N \l_@@_final_i_int }
4234     { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
4235     { } % for the name of the block
4236     }
4237 }
```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analyse of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```
4238 \cs_new_protected:Npn \@@_open_shorten:
4239 {
4240   \bool_if:NT \l_@@_initial_open_bool
4241   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4242   \bool_if:NT \l_@@_final_open_bool
4243   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4244 }
```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row `#1` and column `#2`. As of now, it’s only the whole array (excepted exterior rows and columns).

```
4245 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4246 {
4247   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4248   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4249   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4250   \int_set_eq:NN \l_@@_col_max_int \c@jCol
```

We do a loop over all the submatrices specified in the code-before. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```
4251   \seq_if_empty:NF \g_@@_submatrix_seq
4252   {
4253     \seq_map_inline:Nn \g_@@_submatrix_seq
4254     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4255   }
4256 }
```

`#1` and `#2` are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. `#3`, `#4`, `#5` and `#6` are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programmation of that command with the the standard syntax of L3.

```
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
```

```

}
{
  \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
  \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
  \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
  \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
}
}

```

However, for efficiency, we will use the following version.

```

4257 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4258 {
4259   \if_int_compare:w #3 > #1
4260   \else:
4261     \if_int_compare:w #1 > #5
4262     \else:
4263       \if_int_compare:w #4 > #2
4264       \else:
4265         \if_int_compare:w #2 > #6
4266         \else:
4267           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4268           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4269           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4270           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4271         \fi:
4272       \fi:
4273     \fi:
4274   \fi:
4275 }

4276 \cs_new_protected:Npn \@@_set_initial_coords:
4277 {
4278   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4279   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4280 }
4281 \cs_new_protected:Npn \@@_set_final_coords:
4282 {
4283   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4284   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4285 }
4286 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4287 {
4288   \pgfpointanchor
4289   {
4290     \@@_env:
4291     - \int_use:N \l_@@_initial_i_int
4292     - \int_use:N \l_@@_initial_j_int
4293   }
4294   { #1 }
4295   \@@_set_initial_coords:
4296 }
4297 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4298 {
4299   \pgfpointanchor
4300   {
4301     \@@_env:
4302     - \int_use:N \l_@@_final_i_int
4303     - \int_use:N \l_@@_final_j_int
4304   }
4305   { #1 }
4306   \@@_set_final_coords:
4307 }

```

```

4308 \cs_new_protected:Npn \@@_open_x_initial_dim:
4309 {
4310   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4311   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4312   {
4313     \cs_if_exist:cT
4314     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4315     {
4316       \pgfpointanchor
4317       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4318       { west }
4319       \dim_set:Nn \l_@@_x_initial_dim
4320       { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
4321     }
4322   }

```

If, in fact, all the cells of the column are empty (no PGF/Tikz nodes in those cells).

```

4323   \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
4324   {
4325     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4326     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4327     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4328   }
4329 }

```

```

4330 \cs_new_protected:Npn \@@_open_x_final_dim:
4331 {
4332   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4333   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
4334   {
4335     \cs_if_exist:cT
4336     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4337     {
4338       \pgfpointanchor
4339       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4340       { east }
4341       \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
4342       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4343     }
4344   }

```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```

4345   \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
4346   {
4347     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4348     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4349     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4350   }
4351 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4352 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4353 {
4354   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4355   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4356   {
4357     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4358   \group_begin:
4359   \@@_open_shorten:

```

```

4360     \int_if_zero:nTF { #1 }
4361     { \color { nicematrix-first-row } }
4362     {

```

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

```

4363         \int_compare:nNtT { #1 } = { \l_@@_last_row_int }
4364         { \color { nicematrix-last-row } }
4365     }
4366     \keys_set:nn { nicematrix / xdots } { #3 }
4367     \@@_color:o \l_@@_xdots_color_tl
4368     \@@_actually_draw_Ldots:
4369     \group_end:
4370 }
4371 }

```

The command `\@@_actually_draw_Ldots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

The following function is also used by `\Hdotsfor`.

```

4372 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4373 {
4374     \bool_if:NTF \l_@@_initial_open_bool
4375     {
4376         \@@_open_x_initial_dim:
4377         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4378         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4379     }
4380     { \@@_set_initial_coords_from_anchor:n { base-east } }
4381     \bool_if:NTF \l_@@_final_open_bool
4382     {
4383         \@@_open_x_final_dim:
4384         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4385         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4386     }
4387     { \@@_set_final_coords_from_anchor:n { base-west } }

```

Now the case of a `\Hdotsfor` (or when there is only a `\Ldots`) in the “last row” (that case will probably arise when the final user draws an arrow to indicate the number of columns of the matrix). In the “first row”, we don’t need any adjustment.

```

4388     \bool_lazy_all:nTF
4389     {
4390         \l_@@_initial_open_bool
4391         \l_@@_final_open_bool
4392         { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
4393     }
4394     {
4395         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4396         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4397     }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of text. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4398     {
4399         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim

```

```

4400     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4401   }
4402   \@@_draw_line:
4403 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4404 \cs_new_protected:Npn \@@_draw_Cdots:nmn #1 #2 #3
4405 {
4406   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4407   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4408     {
4409       \@@_find_extremities_of_line:nmmn { #1 } { #2 } { 0 } { 1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4410     \group_begin:
4411     \@@_open_shorten:
4412     \int_if_zero:nTF { #1 }
4413       { \color { nicematrix-first-row } }
4414     {

```

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

```

4415       \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
4416       { \color { nicematrix-last-row } }
4417     }
4418     \keys_set:nn { nicematrix / xdots } { #3 }
4419     \@@_color:o \l_@@_xdots_color_tl
4420     \@@_actually_draw_Cdots:
4421   \group_end:
4422 }
4423 }

```

The command `\@@_actually_draw_Cdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4424 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4425 {
4426   \bool_if:NTF \l_@@_initial_open_bool
4427     { \@@_open_x_initial_dim: }
4428     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4429   \bool_if:NTF \l_@@_final_open_bool
4430     { \@@_open_x_final_dim: }
4431     { \@@_set_final_coords_from_anchor:n { mid-west } }
4432   \bool_lazy_and:nnTF
4433     { \l_@@_initial_open_bool }
4434     { \l_@@_final_open_bool }
4435     {
4436       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4437       \dim_set_eq:NN \l_tmpa_dim \pgf@y
4438       \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4439       \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }

```

```

4440     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4441   }
4442   {
4443     \bool_if:NT \l_@@_initial_open_bool
4444     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4445     \bool_if:NT \l_@@_final_open_bool
4446     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4447   }
4448   \@@_draw_line:
4449 }
4450 \cs_new_protected:Npn \@@_open_y_initial_dim:
4451 {
4452   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4453   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4454   {
4455     \cs_if_exist:cT
4456     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4457     {
4458       \pgfpointanchor
4459       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4460       { north }
4461       \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
4462       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4463     }
4464   }
4465   \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
4466   {
4467     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4468     \dim_set:Nn \l_@@_y_initial_dim
4469     {
4470       \fp_to_dim:n
4471       {
4472         \pgf@y
4473         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4474       }
4475     }
4476   }
4477 }
4478 \cs_new_protected:Npn \@@_open_y_final_dim:
4479 {
4480   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4481   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
4482   {
4483     \cs_if_exist:cT
4484     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4485     {
4486       \pgfpointanchor
4487       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4488       { south }
4489       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
4490       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4491     }
4492   }
4493   \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
4494   {
4495     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4496     \dim_set:Nn \l_@@_y_final_dim
4497     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4498   }
4499 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4500 \cs_new_protected:Npn \@@_draw_Vdots:nmn #1 #2 #3
4501 {
4502   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4503   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4504   {
4505     \@@_find_extremities_of_line:nmmm { #1 } { #2 } { 1 } { 0 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4506     \group_begin:
4507     \@@_open_shorten:
4508     \int_if_zero:nTF { #2 }
4509     { \color { nicematrix-first-col } }
4510     {
4511       \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
4512       { \color { nicematrix-last-col } }
4513     }
4514     \keys_set:nn { nicematrix / xdots } { #3 }
4515     \@@_color:o \l_@@_xdots_color_tl
4516     \bool_if:NTF \l_@@_Vbrace_bool
4517     { \@@_actually_draw_Vbrace: }
4518     { \@@_actually_draw_Vdots: }
4519   \group_end:
4520 }
4521 }

```

The following function is used by regular calls of `\Vdots` or `\Vdotsfor` but not by `\Vbrace`. The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4522 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4523 {
4524   \bool_lazy_and:mnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
4525   { \@@_actually_draw_Vdots_i: }
4526   { \@@_actually_draw_Vdots_ii: }
4527   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4528   \@@_draw_line:
4529 }

```

First, the case of a dotted line open on both sides.

```

4530 \cs_new_protected:Npn \@@_actually_draw_Vdots_i:
4531 {
4532   \@@_open_y_initial_dim:
4533   \@@_open_y_final_dim:
4534   \int_if_zero:nTF { \l_@@_initial_j_int }

```

We have a dotted line open on both sides in the “first column”.

```

4535   {
4536     \@@_qpoint:n { col - 1 }
4537     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4538     \dim_sub:Nn \l_@@_x_initial_dim
4539     { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4540   }
4541   {

```



```

4542 \bool_lazy_and:nnTF
4543 { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
4544 {
4545 \int_compare_p:nNn
4546 { \l_@@_initial_j_int } = { \g_@@_col_total_int }
4547 }

```

We have a dotted line open on both sides and which is in the “last column”.

```

4548 {
4549 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4550 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4551 \dim_add:Nn \l_@@_x_initial_dim
4552 { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4553 }

```

We have a dotted line open on both sides which is *not* in an exterior column.

```

4554 {
4555 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4556 \dim_set_eq:NN \l_tmpa_dim \pgf@x
4557 \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4558 \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4559 }
4560 }
4561 }

```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The main task is to determine the x -value of the dotted line to draw.

The boolean `\l_tmpa_bool` will indicate whether the column is of type `l` or may be considered as if.

```

4562 \cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
4563 {
4564 \bool_set_false:N \l_tmpa_bool
4565 \bool_if:NF \l_@@_initial_open_bool
4566 {
4567 \bool_if:NF \l_@@_final_open_bool
4568 {
4569 \@@_set_initial_coords_from_anchor:n { south-west }
4570 \@@_set_final_coords_from_anchor:n { north-west }
4571 \bool_set:Nn \l_tmpa_bool
4572 {
4573 \dim_compare_p:nNn
4574 { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4575 }
4576 }
4577 }

```

Now, we try to determine whether the column is of type `c` or may be considered as if.

```

4578 \bool_if:NTF \l_@@_initial_open_bool
4579 {
4580 \@@_open_y_initial_dim:
4581 \@@_set_final_coords_from_anchor:n { north }
4582 \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4583 }
4584 {
4585 \@@_set_initial_coords_from_anchor:n { south }
4586 \bool_if:NTF \l_@@_final_open_bool
4587 { \@@_open_y_final_dim: }

```

Now the case where both extremities are closed. The first conditional tests whether the column is of type `c` or may be considered as if.

```

4588 {
4589 \@@_set_final_coords_from_anchor:n { north }
4590 \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
4591 {

```

```

4592         \dim_set:Nn \l_@@_x_initial_dim
4593         {
4594             \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
4595             \l_@@_x_initial_dim \l_@@_x_final_dim
4596         }
4597     }
4598 }
4599 }
4600 }

```

The following function is used by `\Vbrace` but not by regular uses of `\Vdots` or `\Vdotsfor`. The command `\@@_actually_draw_Vbrace:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4601 \cs_new_protected:Npn \@@_actually_draw_Vbrace:
4602 {
4603     \bool_if:NTF \l_@@_initial_open_bool
4604     { \@@_open_y_initial_dim: }
4605     { \@@_set_initial_coords_from_anchor:n { south } }
4606     \bool_if:NTF \l_@@_final_open_bool
4607     { \@@_open_y_final_dim: }
4608     { \@@_set_final_coords_from_anchor:n { north } }

```

Now, we have the correct values for the y -values of both extremities of the brace. We have to compute the x -value (there is only one x -value since, of course, the brace is vertical).

If we are in the first (exterior) column, the brace must be drawn right flush.

```

4609     \int_if_zero:nTF { \l_@@_initial_j_int }
4610     {
4611         \@@_qpoint:n { col - 1 }
4612         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4613         \dim_sub:Nn \l_@@_x_initial_dim
4614         { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
4615     }

```

Elsewhere, the brace must be drawn left flush.

```

4616     {
4617         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4618         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4619         \dim_add:Nn \l_@@_x_initial_dim
4620         { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
4621     }

```

We draw a vertical rule and that's why, of course, both x -values are equal.

```

4622     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4623     \@@_draw_line:
4624 }

```

```

4625 \cs_new:Npn \@@_colsep:
4626 { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }

```

For the diagonal lines, the situation is a bit more complicated because, by default, we parallelize the diagonals lines. The first diagonal line is drawn and then, all the other diagonal lines are drawn parallel to the first one.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```
4627 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4628 {
4629   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4630   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4631   {
4632     \@@_find_extremities_of_line:nmmn { #1 } { #2 } { 1 } { 1 }
```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```
4633   \group_begin:
4634     \@@_open_shorten:
4635     \keys_set:nn { nicematrix / xdots } { #3 }
4636     \@@_color:o \l_@@_xdots_color_tl
4637     \@@_actually_draw_Ddots:
4638   \group_end:
4639 }
4640 }
```

The command `\@@_actually_draw_Ddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```
4641 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4642 {
4643   \bool_if:NTF \l_@@_initial_open_bool
4644   {
4645     \@@_open_y_initial_dim:
4646     \@@_open_x_initial_dim:
4647   }
4648   { \@@_set_initial_coords_from_anchor:n { south-east } }
4649   \bool_if:NTF \l_@@_final_open_bool
4650   {
4651     \@@_open_x_final_dim:
4652     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4653   }
4654   { \@@_set_final_coords_from_anchor:n { north-west } }
```

We have retrieved the coordinates in the usual way (they are stored in `\l_@@_x_initial_dim`, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

```
4655   \bool_if:NT \l_@@_parallelize_diags_bool
4656   {
4657     \int_gincr:N \g_@@_ddots_int
```

We test if the diagonal line is the first one (the counter `\g_@@_ddots_int` is created for this usage).

```
4658     \int_compare:nNnTF { \g_@@_ddots_int } = { \c_one_int }
```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```
4659     {
```

```

4660     \dim_gset:Nn \g_@@_delta_x_one_dim
4661     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4662     \dim_gset:Nn \g_@@_delta_y_one_dim
4663     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4664   }

```

If the diagonal line is not the first one, we have to adjust the second extremity of the line by modifying the coordinate `\l_@@_x_initial_dim`.

```

4665     {
4666       \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
4667       {
4668         \dim_set:Nn \l_@@_y_final_dim
4669         {
4670           \l_@@_y_initial_dim +
4671           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4672           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4673         }
4674       }
4675     }
4676   }
4677 \@@_draw_line:
4678 }

```

We draw the `\Iddots` diagonals in the same way.

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4679 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4680 {
4681   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4682   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4683   {
4684     \@@_find_extremities_of_line:nmmm { #1 } { #2 } { 1 } { -1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4685     \group_begin:
4686     \@@_open_shorten:
4687     \keys_set:nn { nicematrix / xdots } { #3 }
4688     \@@_color:o \l_@@_xdots_color_tl
4689     \@@_actually_draw_Iddots:
4690   \group_end:
4691 }
4692 }

```

The command `\@@_actually_draw_Iddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4693 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4694 {
4695   \bool_if:NTF \l_@@_initial_open_bool
4696   {
4697     \@@_open_y_initial_dim:
4698     \@@_open_x_initial_dim:
4699   }

```

```

4700     { \l_@@_set_initial_coords_from_anchor:n { south-west } }
4701 \bool_if:NTF \l_@@_final_open_bool
4702   {
4703     \l_@@_open_y_final_dim:
4704     \l_@@_open_x_final_dim:
4705   }
4706   { \l_@@_set_final_coords_from_anchor:n { north-east } }
4707 \bool_if:NT \l_@@_parallelize_diags_bool
4708   {
4709     \int_gincr:N \g_@@_iddots_int
4710     \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
4711     {
4712       \dim_gset:Nn \g_@@_delta_x_two_dim
4713       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4714       \dim_gset:Nn \g_@@_delta_y_two_dim
4715       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4716     }
4717     {
4718       \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
4719       {
4720         \dim_set:Nn \l_@@_y_final_dim
4721         {
4722           \l_@@_y_initial_dim +
4723           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4724           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4725         }
4726       }
4727     }
4728   }
4729 \l_@@_draw_line:
4730 }

```

17 The actual instructions for drawing the dotted lines with Tikz

The command `\l_@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

4731 \cs_new_protected:Npn \l_@@_draw_line:
4732   {
4733     \pgfrememberpicturepositiononpagetrue
4734     \pgf@relevantforpicturesizefalse
4735     \bool_lazy_or:nnTF
4736       { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4737       { \l_@@_dotted_bool }
4738       { \l_@@_draw_standard_dotted_line: }
4739       { \l_@@_draw_unstandard_dotted_line: }
4740   }

```

We have to do a special construction with `\exp_args:No` to be able to put in the list of options in the correct place in the Tikz instruction.

```

4741 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4742 {
4743   \begin { scope }
4744   \@@_draw_unstandard_dotted_line:o
4745   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4746 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4747 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4748 {
4749   \@@_draw_unstandard_dotted_line:nooo
4750   { #1 }
4751   \l_@@_xdots_up_tl
4752   \l_@@_xdots_down_tl
4753   \l_@@_xdots_middle_tl
4754 }
4755 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4756 \hook_gput_code:nnn { begindocument } { . }
4757 {
4758   \IfPackageLoadedT { tikz }
4759   {
4760     \tikzset
4761     {
4762       @@_node_above / .style = { sloped , above } ,
4763       @@_node_below / .style = { sloped , below } ,
4764       @@_node_middle / .style =
4765       {
4766         sloped ,
4767         inner-sep = \c_@@_innersep_middle_dim
4768       }
4769     }
4770   }
4771 }

4772 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4773 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shorten >` and `shorten <` of TikZ in the command `\draw`, we don't have the expected output with `{decorate,decoration=brace}` is used.

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4774 \dim_zero_new:N \l_@@_l_dim
4775 \dim_set:Nn \l_@@_l_dim
4776 {
4777   \fp_to_dim:n
4778   {
4779     sqrt
4780     (
4781       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4782       +
4783       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4784     )
4785   }
4786 }

```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the aux file to say that one more compilation should be done.

```

4787   \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4788   {
4789     \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4790     \@@_draw_unstandard_dotted_line_i:
4791   }

```

If the key `xdots/horizontal-labels` has been used.

```

4792   \bool_if:NT \l_@@_xdots_h_labels_bool
4793   {
4794     \tikzset
4795     {
4796       @@_node_above / .style = { auto = left } ,
4797       @@_node_below / .style = { auto = right } ,
4798       @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4799     }
4800   }
4801   \tl_if_empty:nF { #4 }
4802   { \tikzset { @@_node_middle / .append-style = { fill = white } } }
4803   \draw
4804   [ #1 ]
4805   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can't put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

4806     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4807     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4808     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4809     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4810   \end { scope }
4811 }
4812 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4813 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4814 {
4815   \dim_set:Nn \l_tmpa_dim
4816   {
4817     \l_@@_x_initial_dim
4818     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4819     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4820   }
4821   \dim_set:Nn \l_tmpb_dim
4822   {
4823     \l_@@_y_initial_dim
4824     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4825     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4826   }
4827   \dim_set:Nn \l_@@_tmpc_dim
4828   {
4829     \l_@@_x_final_dim
4830     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4831     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4832   }
4833   \dim_set:Nn \l_@@_tmpd_dim
4834   {
4835     \l_@@_y_final_dim
4836     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4837     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4838   }
4839   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4840   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim

```

```

4841 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4842 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4843 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4844 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4845 {
4846 \group_begin:

```

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4847 \dim_zero_new:N \l_@@_l_dim
4848 \dim_set:Nn \l_@@_l_dim
4849 {
4850 \fp_to_dim:n
4851 {
4852 sqrt
4853 (
4854 (\l_@@_x_final_dim - \l_@@_x_initial_dim) ^ 2
4855 +
4856 (\l_@@_y_final_dim - \l_@@_y_initial_dim) ^ 2
4857 )
4858 }
4859 }

```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the aux file to say that one more compilation should be done.

```

4860 \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
4861 {
4862 \dim_compare:nNnT { \l_@@_l_dim } > { 1 pt }
4863 { \@@_draw_standard_dotted_line_i: }
4864 }
4865 \group_end:
4866 \bool_lazy_all:nF
4867 {
4868 { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4869 { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4870 { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4871 }
4872 { \@@_labels_standard_dotted_line: }
4873 }
4874 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4875 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4876 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4877 \int_set:Nn \l_tmpa_int
4878 {
4879 \dim_ratio:nn
4880 {
4881 \l_@@_l_dim
4882 - \l_@@_xdots_shorten_start_dim
4883 - \l_@@_xdots_shorten_end_dim
4884 }
4885 { \l_@@_xdots_inter_dim }
4886 }

```


The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4887   \dim_set:Nn \l_tmpa_dim
4888   {
4889     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4890     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4891   }
4892   \dim_set:Nn \l_tmpb_dim
4893   {
4894     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4895     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4896   }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4897   \dim_gadd:Nn \l_@@_x_initial_dim
4898   {
4899     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4900     \dim_ratio:nn
4901     {
4902       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4903       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4904     }
4905     { 2 \l_@@_l_dim }
4906   }
4907   \dim_gadd:Nn \l_@@_y_initial_dim
4908   {
4909     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4910     \dim_ratio:nn
4911     {
4912       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4913       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4914     }
4915     { 2 \l_@@_l_dim }
4916   }
4917   \pgf@relevantforpicturesizefalse
4918   \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
4919   {
4920     \pgfpathcircle
4921     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4922     { \l_@@_xdots_radius_dim }
4923     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4924     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4925   }
4926   \pgfusepathqfill
4927 }

```

```

4928 \cs_new_protected:Npn \@@_labels_standard_dotted_line:
4929 {
4930   \pgfscope
4931   \pgftransformshift
4932   {
4933     \pgfpointlineattime { 0.5 }
4934     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4935     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4936   }
4937   \fp_set:Nn \l_tmpa_fp
4938   {
4939     atand
4940     (
4941       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4942       \l_@@_x_final_dim - \l_@@_x_initial_dim
4943     )

```

```

4944 }
4945 \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4946 \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4947 \tl_if_empty:NF \l_@@_xdots_middle_tl
4948 {
4949   \begin { pgfscope }
4950   \pgfset { inner-sep = \c_@@_innersep_middle_dim }
4951   \pgfnode
4952     { rectangle }
4953     { center }
4954     {
4955       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4956       {
4957         \c_math_toggle_token
4958         \scriptstyle \l_@@_xdots_middle_tl
4959         \c_math_toggle_token
4960       }
4961     }
4962     { }
4963     {
4964       \pgfsetfillcolor { white }
4965       \pgfusepath { fill }
4966     }
4967   \end { pgfscope }
4968 }
4969 \tl_if_empty:NF \l_@@_xdots_up_tl
4970 {
4971   \pgfnode
4972     { rectangle }
4973     { south }
4974     {
4975       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4976       {
4977         \c_math_toggle_token
4978         \scriptstyle \l_@@_xdots_up_tl
4979         \c_math_toggle_token
4980       }
4981     }
4982     { }
4983     { \pgfusepath { } }
4984 }
4985 \tl_if_empty:NF \l_@@_xdots_down_tl
4986 {
4987   \pgfnode
4988     { rectangle }
4989     { north }
4990     {
4991       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4992       {
4993         \c_math_toggle_token
4994         \scriptstyle \l_@@_xdots_down_tl
4995         \c_math_toggle_token
4996       }
4997     }
4998     { }
4999     { \pgfusepath { } }
5000 }
5001 \endpgfscope
5002 }

```

18 User commands available in the new environments

The commands `\@@_Ldots:`, `\@@_Cdots:`, `\@@_Vdots:`, `\@@_Ddots:` and `\@@_Iddots:` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use underscore, and, in that case, the catcode is 13 because underscore activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```
5003 \hook_gput_code:nnn { begindocument } { . }
5004 {
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
5005 \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m E { _ ^ : } { { } { } { } } }
5006 \cs_new_protected:Npn \@@_Ldots:
5007 { \@@_collect_options:n { \@@_Ldots_i } }
5008 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
5009 {
5010 \int_if_zero:nTF { \c@jCol }
5011 { \@@_error:nn { in~first~col } { \Ldots } }
5012 {
5013 \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5014 { \@@_error:nn { in~last~col } { \Ldots } }
5015 {
5016 \@@_instruction_of_type:nnn { \c_false_bool } { Ldots }
5017 { #1 , down = #2 , up = #3 , middle = #4 }
5018 }
5019 }
5020 \bool_if:NF \l_@@_nullify_dots_bool
5021 { \phantom { \ensuremath { \@@_old_ldots: } } } }
5022 \bool_gset_true:N \g_@@_empty_cell_bool
5023 }
```

```
5024 \cs_new_protected:Npn \@@_Cdots:
5025 { \@@_collect_options:n { \@@_Cdots_i } }
5026 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5027 {
5028 \int_if_zero:nTF { \c@jCol }
5029 { \@@_error:nn { in~first~col } { \Cdots } }
5030 {
5031 \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
5032 { \@@_error:nn { in~last~col } { \Cdots } }
5033 {
5034 \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
5035 { #1 , down = #2 , up = #3 , middle = #4 }
5036 }
5037 }
5038 \bool_if:NF \l_@@_nullify_dots_bool
5039 { \phantom { \ensuremath { \@@_old_cdots: } } } }
5040 \bool_gset_true:N \g_@@_empty_cell_bool
5041 }
```

```
5042 \cs_new_protected:Npn \@@_Vdots:
5043 { \@@_collect_options:n { \@@_Vdots_i } }
5044 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5045 {
5046 \int_if_zero:nTF { \c@iRow }
```

```

5047     { \@@_error:nn { in~first~row } { \Vdots } }
5048     {
5049         \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
5050         { \@@_error:nn { in~last~row } { \Vdots } }
5051         {
5052             \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
5053             { #1 , down = #2 , up = #3 , middle = #4 }
5054         }
5055     }
5056     \bool_if:NF \l_@@_nullify_dots_bool
5057     { \phantom { \ensuremath { \@@_old_vdots: } } } }
5058     \bool_gset_true:N \g_@@_empty_cell_bool
5059 }

5060 \cs_new_protected:Npn \@@_Ddots:
5061 { \@@_collect_options:n { \@@_Ddots_i } }
5062 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5063 {
5064     \int_case:nnF \c@iRow
5065     {
5066         0 { \@@_error:nn { in~first~row } { \Ddots } }
5067         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
5068     }
5069     {
5070         \int_case:nnF \c@jCol
5071         {
5072             0 { \@@_error:nn { in~first~col } { \Ddots } }
5073             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
5074         }
5075         {
5076             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5077             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5078             { #1 , down = #2 , up = #3 , middle = #4 }
5079         }
5080     }
5081 }
5082 \bool_if:NF \l_@@_nullify_dots_bool
5083 { \phantom { \ensuremath { \@@_old_ddots: } } } }
5084 \bool_gset_true:N \g_@@_empty_cell_bool
5085 }

5086 \cs_new_protected:Npn \@@_Iddots:
5087 { \@@_collect_options:n { \@@_Iddots_i } }
5088 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5089 {
5090     \int_case:nnF \c@iRow
5091     {
5092         0 { \@@_error:nn { in~first~row } { \Iddots } }
5093         \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
5094     }
5095     {
5096         \int_case:nnF \c@jCol
5097         {
5098             0 { \@@_error:nn { in~first~col } { \Iddots } }
5099             \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
5100         }
5101         {
5102             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5103             \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
5104             { #1 , down = #2 , up = #3 , middle = #4 }
5105         }
5106     }

```

```

5107     \bool_if:NF \l_@@_nullify_dots_bool
5108     { \phantom { \ensuremath { \@@_old_iddots: } } }
5109     \bool_gset_true:N \g_@@_empty_cell_bool
5110   }
5111 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5112 \keys_define:nn { nicematrix / Ddots }
5113 {
5114   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5115   draw-first .default:n = true ,
5116   draw-first .value_forbidden:n = true
5117 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5118 \cs_new_protected:Npn \@@_Hspace:
5119 {
5120   \bool_gset_true:N \g_@@_empty_cell_bool
5121   \hspace
5122 }

```

In the environments of `nicematrix`, the command `\multicolumn` is redefined. We will patch the environment `{tabular}` to go back to the previous value of `\multicolumn`.

```

5123 \cs_set_eq:NN \@@_old_multicolumn: \multicolumn

```

The command `\@@_Hdotsfor` will be linked to `\Hdotsfor` in `{NiceArrayWithDelims}`. Tikz nodes are created also in the implicit cells of the `\Hdotsfor` (maybe we should modify that point).

This command must *not* be protected since it begins with `\multicolumn`.

```

5124 \cs_new:Npn \@@_Hdotsfor:
5125 {
5126   \bool_lazy_and:nnTF
5127   { \int_if_zero_p:n { \c@jCol } }
5128   { \int_if_zero_p:n { \l_@@_first_col_int } }
5129   {
5130     \bool_if:NTF \g_@@_after_col_zero_bool
5131     {
5132       \multicolumn { 1 } { c } { }
5133       \@@_Hdotsfor_i:
5134     }
5135     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5136   }
5137   {
5138     \multicolumn { 1 } { c } { }
5139     \@@_Hdotsfor_i:
5140   }
5141 }

```

The command `\@@_Hdotsfor_i:` is defined with `\NewDocumentCommand` because it has an optional argument. Note that such a command defined by `\NewDocumentCommand` is protected and that's why we have put the `\multicolumn` before (in the definition of `\@@_Hdotsfor:`).

```

5142 \hook_gput_code:nnn { begindocument } { . }
5143 {

```

We don't put `!` before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5144   \cs_new_protected:Npn \@@_Hdotsfor_i:
5145   { \@@_collect_options:n { \@@_Hdotsfor_ii } }

```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```

5146 \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5147 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
5148 {
5149   \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5150   {
5151     \@@_Hdotsfor:nnnn
5152     { \int_use:N \c@iRow }
5153     { \int_use:N \c@jCol }
5154     { #2 }
5155     {
5156       #1 , #3 ,
5157       down = \exp_not:n { #4 } ,
5158       up = \exp_not:n { #5 } ,
5159       middle = \exp_not:n { #6 }
5160     }
5161   }
5162   \prg_replicate:nn { #2 - 1 }
5163   {
5164     &
5165     \multicolumn { 1 } { c } { }
5166     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5167   }
5168 }
5169 }

```

```

5170 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5171 {
5172   \bool_set_false:N \l_@@_initial_open_bool
5173   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5174 \int_set:Nn \l_@@_initial_i_int { #1 }
5175 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5176 \int_compare:nNnTF { #2 } = { \c_one_int }
5177 {
5178   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5179   \bool_set_true:N \l_@@_initial_open_bool
5180 }
5181 {
5182   \cs_if_exist:cTF
5183   {
5184     pgf @ sh @ ns @ \@@_env:
5185     - \int_use:N \l_@@_initial_i_int
5186     - \int_eval:n { #2 - 1 }
5187   }
5188   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5189   {
5190     \int_set:Nn \l_@@_initial_j_int { #2 }
5191     \bool_set_true:N \l_@@_initial_open_bool
5192   }
5193 }
5194 \int_compare:nNnTF { #2 + #3 - 1 } = { \c@jCol }
5195 {
5196   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5197   \bool_set_true:N \l_@@_final_open_bool
5198 }
5199 {
5200   \cs_if_exist:cTF
5201   {
5202     pgf @ sh @ ns @ \@@_env:

```

```

5203     - \int_use:N \l_@@_final_i_int
5204     - \int_eval:n { #2 + #3 }
5205   }
5206   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5207   {
5208     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5209     \bool_set_true:N \l_@@_final_open_bool
5210   }
5211 }
5212 \group_begin:
5213 \@@_open_shorten:
5214 \int_if_zero:nTF { #1 }
5215   { \color { nicematrix-first-row } }
5216   {
5217     \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
5218     { \color { nicematrix-last-row } }
5219   }
5220 \keys_set:nn { nicematrix / xdots } { #4 }
5221 \@@_color:o \l_@@_xdots_color_tl
5222 \@@_actually_draw_Ldots:
5223 \group_end:

```

We declare all the cells concerned by the `\Hdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5224   \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5225   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5226 }

```

```

5227 \hook_gput_code:nnn { begindocument } { . }
5228 {
5229   \cs_new_protected:Npn \@@_Vdotsfor:
5230     { \@@_collect_options:n { \@@_Vdotsfor_i } }

```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that’s why we are in a `\AtBeginDocument`).

```

5231   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m 0 { } E { _ ^ : } { { } { } { } } }
5232   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
5233   {
5234     \bool_gset_true:N \g_@@_empty_cell_bool
5235     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5236     {
5237       \@@_Vdotsfor:nnnn
5238       { \int_use:N \c@iRow }
5239       { \int_use:N \c@jCol }
5240       { #2 }
5241       {
5242         #1 , #3 ,
5243         down = \exp_not:n { #4 } ,
5244         up = \exp_not:n { #5 } ,
5245         middle = \exp_not:n { #6 }
5246       }
5247     }
5248   }
5249 }

```

#1 is the number of row;

#2 is the number of column;

#3 is the numbers of rows which are involved;

```

5250 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5251 {
5252   \bool_set_false:N \l_@@_initial_open_bool
5253   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
5254 \int_set:Nn \l_@@_initial_j_int { #2 }
5255 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

For the row, it's a bit more complicated.

```
5256 \int_compare:nNnTF { #1 } = { \c_one_int }
5257 {
5258   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5259   \bool_set_true:N \l_@@_initial_open_bool
5260 }
5261 {
5262   \cs_if_exist:cTF
5263   {
5264     pgf @ sh @ ns @ \@@_env:
5265     - \int_eval:n { #1 - 1 }
5266     - \int_use:N \l_@@_initial_j_int
5267   }
5268   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5269   {
5270     \int_set:Nn \l_@@_initial_i_int { #1 }
5271     \bool_set_true:N \l_@@_initial_open_bool
5272   }
5273 }
5274 \int_compare:nNnTF { #1 + #3 - 1 } = { \c_iRow }
5275 {
5276   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5277   \bool_set_true:N \l_@@_final_open_bool
5278 }
5279 {
5280   \cs_if_exist:cTF
5281   {
5282     pgf @ sh @ ns @ \@@_env:
5283     - \int_eval:n { #1 + #3 }
5284     - \int_use:N \l_@@_final_j_int
5285   }
5286   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5287   {
5288     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5289     \bool_set_true:N \l_@@_final_open_bool
5290   }
5291 }
5292 \group_begin:
5293 \@@_open_shorten:
5294 \int_if_zero:nTF { #2 }
5295 { \color { nicematrix-first-col } }
5296 {
5297   \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
5298   { \color { nicematrix-last-col } }
5299 }
5300 \keys_set:mn { nicematrix / xdots } { #4 }
5301 \@@_color:o \l_@@_xdots_color_tl
5302 \bool_if:NTF \l_@@_Vbrace_bool
5303 { \@@_actually_draw_Vbrace: }
5304 { \@@_actually_draw_Vdots: }
5305 \group_end:
```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```
5306 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5307 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5308 }
```


The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5309 \NewDocumentCommand \@@_rotate: { 0 { } }
5310 {
5311   \bool_gset_true:N \g_@@_rotate_bool
5312   \keys_set:nn { nicematrix / rotate } { #1 }
5313   \ignorespaces
5314 }

5315 \keys_define:nn { nicematrix / rotate }
5316 {
5317   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5318   c .value_forbidden:n = true ,
5319   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5320 }

```

19 The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format i - j , our command applies the command `\int_eval:n` to i and j ;
- If not (that is to say, when it's a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹⁴

```

5321 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5322 {
5323   \tl_if_empty:nTF { #2 }
5324     { #1 }
5325     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5326 }
5327 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5328 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

5329 \hook_gput_code:nnn { begindocument } { . }
5330 {

```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```

5331   \tl_set_rescan:Nnn \l_tmpa_tl { }
5332   { 0 { } m ! 0 { } E { _ ^ : } { { } { } { } } }
5333   \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
5334   {
5335     \group_begin:
5336     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5337     \@@_color:o \l_@@_xdots_color_tl
5338     \use:e
5339     {

```

¹⁴Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5340     \@@_line_i:nn
5341     { \@@_double_int_eval:n #2 - \q_stop }
5342     { \@@_double_int_eval:n #3 - \q_stop }
5343   }
5344   \group_end:
5345 }
5346 }

5347 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5348 {
5349   \bool_set_false:N \l_@@_initial_open_bool
5350   \bool_set_false:N \l_@@_final_open_bool
5351   \bool_lazy_or:nnTF
5352   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5353   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5354   { \@@_error:nnn { unknown-cell-for-line-in~CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5355   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5356 }

5357 \hook_gput_code:nnn { begindocument } { . }
5358 {
5359   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5360   {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible” and that why we do this static construction of the command `\@@_draw_line_ii:`.

```

5361     \c_@@_pgfortikzpicture_tl
5362     \@@_draw_line_iii:nn { #1 } { #2 }
5363     \c_@@_endpgfortikzpicture_tl
5364   }
5365 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5366 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5367 {
5368   \pgfrememberpicturepositiononpagetrue
5369   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5370   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5371   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5372   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5373   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5374   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5375   \@@_draw_line:
5376 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

However, both arguments are implicit because they are taken by curryfication.

```
5377 \cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
5378 \cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5379 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5380 {
5381   \tl_gput_right:Ne \g_@@_row_style_tl
5382   {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5383   \exp_not:N
5384   \@@_if_row_less_than:nn
5385   { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
```

The `\scan_stop:` is mandatory (for ex. for the case where `\rotate` is used in the argument of `\RowStyle`).

```
5386   {
5387     \exp_not:N
5388     \@@_if_col_greater_than:nn
5389     { \int_eval:n { \c@jCol } }
5390     { \exp_not:n { #1 } \scan_stop: }
5391   }
5392 }
5393 }
5394 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
5395 \keys_define:nn { nicematrix / RowStyle }
5396 {
5397   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5398   cell-space-top-limit .value_required:n = true ,
5399   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5400   cell-space-bottom-limit .value_required:n = true ,
5401   cell-space-limits .meta:n =
5402   {
5403     cell-space-top-limit = #1 ,
5404     cell-space-bottom-limit = #1 ,
5405   } ,
5406   color .tl_set:N = \l_@@_color_tl ,
5407   color .value_required:n = true ,
5408   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5409   bold .default:n = true ,
5410   nb-rows .code:n =
5411   \str_if_eq:eeTF { #1 } { * }
5412   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5413   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5414   nb-rows .value_required:n = true ,
5415   fill .tl_set:N = \l_@@_fill_tl ,
5416   fill .value_required:n = true ,
5417   opacity .tl_set:N = \l_@@_opacity_tl ,
5418   opacity .value_required:n = true ,
5419   rowcolor .tl_set:N = \l_@@_fill_tl ,
5420   rowcolor .value_required:n = true ,
5421   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5422   rounded-corners .default:n = 4 pt ,
5423   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5424 }
```

```

5425 \NewDocumentCommand \l_@@_RowStyle:n { 0 { } m }
5426 {
5427   \group_begin:
5428   \tl_clear:N \l_@@_fill_tl
5429   \tl_clear:N \l_@@_opacity_tl
5430   \tl_clear:N \l_@@_color_tl
5431   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5432   \dim_zero:N \l_@@_rounded_corners_dim
5433   \dim_zero:N \l_tmpa_dim
5434   \dim_zero:N \l_tmpb_dim
5435   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key fill (or its alias rowcolor) has been used.

```

5436   \tl_if_empty:NF \l_@@_fill_tl
5437   {
5438     \l_@@_add_opacity_to_fill:
5439     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5440     {

```

The command \l_@@_exp_color_arg:No is *fully expandable*.

```

5441       \l_@@_exp_color_arg:No \l_@@_roundedrectanglecolor \l_@@_fill_tl
5442       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5443       {
5444         \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5445         - *
5446       }
5447       { \dim_use:N \l_@@_rounded_corners_dim }
5448     }
5449   }
5450   \l_@@_put_in_row_style:n { \exp_not:n { #2 } }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

5451   \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
5452   {
5453     \l_@@_put_in_row_style:e
5454     {
5455       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5456       {

```

It's not possible to change the following code by using \dim_set_eq:NN (because of expansion).

```

5457         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5458         { \dim_use:N \l_tmpa_dim }
5459       }
5460     }
5461   }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

5462   \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
5463   {
5464     \l_@@_put_in_row_style:e
5465     {
5466       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5467       {
5468         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5469         { \dim_use:N \l_tmpb_dim }
5470       }
5471     }
5472   }

```

\l_@@_color_tl is the value of the key color of \RowStyle.

```

5473   \tl_if_empty:NF \l_@@_color_tl
5474   {
5475     \l_@@_put_in_row_style:e
5476     {
5477       \mode_leave_vertical:
5478       \l_@@_color:n { \l_@@_color_tl }

```

```

5479     }
5480   }
\l_@@_bold_row_style_bool is the value of the key bold.
5481   \bool_if:NT \l_@@_bold_row_style_bool
5482   {
5483     \@@_put_in_row_style:n
5484     {
5485       \exp_not:n
5486       {
5487         \if_mode_math:
5488           \c_math_toggle_token
5489           \bfseries \boldmath
5490           \c_math_toggle_token
5491         \else:
5492           \bfseries \boldmath
5493         \fi:
5494       }
5495     }
5496   }
5497   \group_end:
5498   \g_@@_row_style_tl
5499   \ignorespaces
5500 }

```

The following commande must *not* be protected.

```

5501 \cs_new:Npn \@@_rounded_from_row:n #1
5502 {
5503   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “- 1” is *not* a subtraction.

```

5504   { \int_eval:n { #1 } - 1 }
5505   {
5506     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5507     - \exp_not:n { \int_use:N \c@jCol }
5508   }
5509   { \dim_use:N \l_@@_rounded_corners_dim }
5510 }

```

21 Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That’s why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`—in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor`, `\@@_rectanglecolor` and `\@@_rowlistcolors` don’t directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\g_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\g_@@_color_i_tl`. In that token list, the instructions will be written using `\@@_cartesian_color:nn` and `\@@_rectanglecolor:nn`.

`#1` is the color and `#2` is an instruction using that color. Despite its name, the command `\@@_add_to_colors_seq:nn` doesn’t only add a color to `\g_@@_colors_seq`: it also updates the

corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```
5511 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5512 {
```

First, we look for the number of the color and, if it's found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```
5513 \int_zero:N \l_tmpa_int
```

We don't take into account the colors like `myserie!!+` because those colors are special color from a `\definecolorseries` of `xcolor`. `\str_if_in:nnF` is mandatory: don't use `\tl_if_in:nnF`.

```
5514 \str_if_in:nnF { #1 } { !! }
5515 {
5516 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5517 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5518 }
5519 \int_if_zero:nTF { \l_tmpa_int }
```

First, the case where the color is a *new* color (not in the sequence).

```
5520 {
5521 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5522 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5523 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5524 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5525 }
5526 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5527 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5528 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5529 {
5530 \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
5531 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5532 \group_begin:
5533 \pgfsetcornersarced
5534 {
5535 \pgfpoint
5536 { \l_@@_tab_rounded_corners_dim }
5537 { \l_@@_tab_rounded_corners_dim }
5538 }
```

Because we want `nicematrix` compatible with arrays constructed by `array`, the nodes for the rows and columns (that is to say the nodes `row-i` and `col-j`) have not always the expected position, that is to say, there is sometimes a slight shifting of something such as `\arrayrulewidth`. Now, for the clipping, we have to change slightly the position of that clipping whether a rounded rectangle around the array is required. That's the point which is tested in the following line.

```
5539 \bool_if:NTF \l_@@_hvlines_bool
5540 {
5541 \pgfpathrectanglecorners
5542 {
5543 \pgfpointadd
5544 { \@@_qpoint:n { row-1 } }
5545 { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5546 }
5547 {
5548 \pgfpointadd
5549 {
```

```

5550         \@@_qpoint:n
5551         { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5552     }
5553     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5554 }
5555 }
5556 {
5557     \pgfpathrectanglecorners
5558     { \@@_qpoint:n { row-1 } }
5559     {
5560         \pgfpointadd
5561         {
5562             \@@_qpoint:n
5563             { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
5564         }
5565         { \pgfpoint \c_zero_dim \arrayrulewidth }
5566     }
5567 }
5568 \pgfusepath { clip }
5569 \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5570     }
5571 }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

5572 \cs_new_protected:Npn \@@_actually_color:
5573 {
5574     \pgfpicture
5575     \pgf@relevantforpicturesizefalse

```

If the final user has used the key `rounded-corners` for the environment `{NiceTabular}`, we will clip to a rectangle with rounded corners before filling the rectangles.

```

5576     \@@_clip_with_rounded_corners:
5577     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5578     {
5579         \int_compare:nNnTF { ##1 } = { \c_one_int }
5580         {
5581             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5582             \use:c { g_@@_color _ 1 _tl }
5583             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5584         }
5585         {
5586             \begin { pgfscope }
5587                 \@@_color_opacity: ##2
5588                 \use:c { g_@@_color _ ##1 _tl }
5589                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5590                 \pgfusepath { fill }
5591             \end { pgfscope }
5592         }
5593     }
5594 \endpgfpicture
5595 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5596 \cs_new_protected:Npn \@@_color_opacity:
5597 {
5598     \peek_meaning:NTF [
5599         { \@@_color_opacity:w }
5600         { \@@_color_opacity:w [ ] }
5601     }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```
5602 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5603 {
5604   \tl_clear:N \l_tmpa_tl
5605   \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl
\l_tmpa_tl (if not empty) is now the opacity and \l_tmpb_tl (if not empty) is now the colorimetric
space.
```

```
5606   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5607   \tl_if_empty:NTF \l_tmpb_tl
5608     { \@declaredcolor }
5609     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5610 }
```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```
5611 \keys_define:nn { nicematrix / color-opacity }
5612 {
5613   opacity .tl_set:N          = \l_tmpa_tl ,
5614   opacity .value_required:n = true
5615 }
```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
5616 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5617 {
5618   \def \l_@@_rows_tl { #1 }
5619   \def \l_@@_cols_tl { #2 }
5620   \@@_cartesian_path:
5621 }
```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
5622 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5623 {
5624   \tl_if_blank:nF { #2 }
5625   {
5626     \@@_add_to_colors_seq:en
5627     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5628     { \@@_cartesian_color:nn { #3 } { - } }
5629   }
5630 }
```

Here an example: `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```
5631 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5632 {
5633   \tl_if_blank:nF { #2 }
5634   {
5635     \@@_add_to_colors_seq:en
5636     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5637     { \@@_cartesian_color:nn { - } { #3 } }
5638   }
5639 }
```

Here is an example: `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
5640 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5641 {
5642   \tl_if_blank:nF { #2 }
5643   {
5644     \@@_add_to_colors_seq:en
5645     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5646     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5647   }
5648 }
```


The last argument is the radius of the corners of the rectangle.

```

5649 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5650 {
5651   \tl_if_blank:nF { #2 }
5652   {
5653     \@@_add_to_colors_seq:en
5654     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5655     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5656   }
5657 }

```

The last argument is the radius of the corners of the rectangle.

```

5658 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5659 {
5660   \@@_cut_on_hyphen:w #1 \q_stop
5661   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5662   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5663   \@@_cut_on_hyphen:w #2 \q_stop
5664   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5665   \tl_set:Ne \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

5666   \@@_cartesian_path:n { #3 }
5667 }

```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5668 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5669 {
5670   \clist_map_inline:nn { #3 }
5671   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5672 }

5673 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5674 {
5675   \int_step_inline:nn { \c@iRow }
5676   {
5677     \int_step_inline:nn { \c@jCol }
5678     {
5679       \int_if_even:nTF { #####1 + ##1 }
5680       { \@@_cellcolor [ #1 ] { #2 } }
5681       { \@@_cellcolor [ #1 ] { #3 } }
5682       { ##1 - #####1 }
5683     }
5684   }
5685 }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the “corners”.

```

5686 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5687 {
5688   \@@_rectanglecolor [ #1 ] { #2 }
5689   { 1 - 1 }
5690   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5691 }

5692 \keys_define:nn { nicematrix / rowcolors }
5693 {
5694   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,

```

```

5695     respect-blocks .default:n = true ,
5696     cols .tl_set:N = \l_@@_cols_tl ,
5697     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5698     restart .default:n = true ,
5699     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5700 }

```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`.

Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

In `nicematrix`, the command `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`. **#1** (optional) is the color space; **#2** is a list of intervals of rows; **#3** is the list of colors; **#4** is for the optional list of pairs *key=value*.

```

5701 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5702 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5703     \group_begin:
5704     \seq_clear_new:N \l_@@_colors_seq
5705     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5706     \tl_clear_new:N \l_@@_cols_tl
5707     \tl_set:Nn \l_@@_cols_tl { - }
5708     \keys_set:nm { nicematrix / rowcolors } { #4 }

```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```

5709     \int_zero_new:N \l_@@_color_int
5710     \int_set_eq:NN \l_@@_color_int \c_one_int
5711     \bool_if:NT \l_@@_respect_blocks_bool
5712     {

```

We don't want to take into account a block which is completely in the "first column" (number 0) or in the "last column" and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```

5713         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5714         \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5715         { \@@_not_in_exterior_p:nnnn ##1 }
5716     }
5717     \pgfpicture
5718     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5719     \clist_map_inline:nm { #2 }
5720     {
5721         \tl_set:Nn \l_tmpa_tl { ##1 }
5722         \tl_if_in:NnTF \l_tmpa_tl { - }
5723         { \@@_cut_on_hyphen:w ##1 \q_stop }
5724         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, `l_tmpa_tl` and `l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

5725         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5726         \int_set:Nn \l_@@_color_int
5727         { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
5728         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5729         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5730         {

```

We will compute in `\l_tmpb_int` the last row of the "block".

```

5731         \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

5732     \bool_if:NT \l_@@_respect_blocks_bool
5733     {
5734         \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5735         { \@@_intersect_our_row_p:nnnnn #####1 }
5736         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5737     }
5738     \tl_set:Ne \l_@@_rows_tl
5739     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5740     \tl_set:Ne \l_@@_color_tl
5741     {
5742         \@@_color_index:n
5743         {
5744             \int_mod:nn
5745             { \l_@@_color_int - 1 }
5746             { \seq_count:N \l_@@_colors_seq }
5747             + 1
5748         }
5749     }
5750     \tl_if_empty:NF \l_@@_color_tl
5751     {
5752         \@@_add_to_colors_seq:ee
5753         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5754         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5755     }
5756     \int_incr:N \l_@@_color_int
5757     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5758 }
5759 }
5760 \endpgfpicture
5761 \group_end:
5762 }

```

The command `\@@_color_index:n` peeks in `\l_@@_colors_seq` the color at the index `#1`. However, if that color is the symbol `=`, the previous one is poken. This macro is recursive.

```

5763 \cs_new:Npn \@@_color_index:n #1
5764 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5765     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5766     { \@@_color_index:n { #1 - 1 } }
5767     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5768 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5769 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5770 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5771 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5772 {
5773     \int_compare:nNnT { #3 } > { \l_tmpb_int }
5774     { \int_set:Nn \l_tmpb_int { #3 } }
5775 }

```

```

5776 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
5777 {
5778   \int_if_zero:nTF { #4 }
5779   { \prg_return_false: }
5780   {
5781     \int_compare:nNnTF { #2 } > { \c@jCol }
5782     { \prg_return_false: }
5783     { \prg_return_true: }
5784   }
5785 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5786 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
5787 {
5788   \int_compare:nNnTF { #1 } > { \l_tmpa_int }
5789   { \prg_return_false: }
5790   {
5791     \int_compare:nNnTF { \l_tmpa_int } > { #3 }
5792     { \prg_return_false: }
5793     { \prg_return_true: }
5794   }
5795 }

```

The following command uses two implicit arguments: `\l_@@_rows_tl` and `\l_@@_cols_tl` which are specifications for a set of rows and a set of columns. It creates a path but does *not* fill it. It must be filled by another command after. The argument is the radius of the corners. We define below a command `\@@_cartesian_path:` which corresponds to a value 0 pt for the radius of the corners. This command is, in particular, used in `\@@_rectanglecolor:nnn` (used in `\@@_rectanglecolor`, itself used in `\@@_cellcolor`).

```

5796 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5797 {
5798   \dim_compare:nNnTF { #1 } = { \c_zero_dim }
5799   {
5800     \bool_if:NTF \l_@@_nocolor_used_bool
5801     { \@@_cartesian_path_normal_ii: }
5802     {
5803       \clist_if_empty:NTF \l_@@_corners_cells_clist
5804       { \@@_cartesian_path_normal_i:n { #1 } }
5805       { \@@_cartesian_path_normal_ii: }
5806     }
5807   }
5808   { \@@_cartesian_path_normal_i:n { #1 } }
5809 }

```

First, the situation where is a rectangular zone of cells will be colored as a whole (in the instructions of the resulting PDF). The argument is the radius of the corners.

```

5810 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5811 {
5812   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5813   \clist_map_inline:Nn \l_@@_cols_tl
5814   {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5815     \def \l_tmpa_tl { ##1 }
5816     \tl_if_in:NnTF \l_tmpa_tl { - }
5817     { \@@_cut_on_hyphen:w ##1 \q_stop }
5818     { \def \l_tmpb_tl { ##1 } } % 2025-04-16
5819     \tl_if_empty:NTF \l_tmpa_tl
5820     { \def \l_tmpa_tl { 1 } }
5821     {

```

```

5822     \str_if_eq:eeT \l_tmpa_tl { * }
5823     { \def \l_tmpa_tl { 1 } }
5824   }
5825   \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
5826   { \@@_error:n { Invalid~col~number } }
5827   \tl_if_empty:NTF \l_tmpb_tl
5828   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5829   {
5830     \str_if_eq:eeT \l_tmpb_tl { * }
5831     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5832   }
5833   \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
5834   { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5835   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5836   \@@_qpoint:n { col - \l_tmpa_tl }
5837   \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
5838   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5839   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5840   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5841   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows. We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5842   \clist_map_inline:Nn \l_@@_rows_tl
5843   {
5844     \def \l_tmpa_tl { #####1 }
5845     \tl_if_in:NnTF \l_tmpa_tl { - }
5846     { \@@_cut_on_hyphen:w #####1 \q_stop }
5847     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5848     \tl_if_empty:NTF \l_tmpa_tl
5849     { \def \l_tmpa_tl { 1 } }
5850     {
5851       \str_if_eq:eeT \l_tmpa_tl { * }
5852       { \def \l_tmpa_tl { 1 } }
5853     }
5854     \tl_if_empty:NTF \l_tmpb_tl
5855     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5856     {
5857       \str_if_eq:eeT \l_tmpb_tl { * }
5858       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5859     }
5860     \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
5861     { \@@_error:n { Invalid~row~number } }
5862     \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
5863     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_row_total_int } }

```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```

5864   \cs_if_exist:cF
5865   { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5866   {
5867     \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5868     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5869     \@@_qpoint:n { row - \l_tmpa_tl }
5870     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5871     \pgfpathrectanglecorners
5872     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5873     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5874   }
5875 }
5876 }
5877 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5878 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5879 {
5880   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5881   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5882   \clist_map_inline:Nn \l_@@_cols_tl
5883   {
5884     \@@_qpoint:n { col - ##1 }
5885     \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
5886       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5887       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5888     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5889     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5890     \clist_map_inline:Nn \l_@@_rows_tl
5891     {
5892       \@@_if_in_corner:nF { #####1 - ##1 }
5893       {
5894         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5895         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5896         \@@_qpoint:n { row - #####1 }
5897         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5898         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5899         {
5900           \pgfpathrectanglecorners
5901             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5902             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5903         }
5904       }
5905     }
5906   }
5907 }

```

The following command corresponds to a radius of the corners equal to 0 pt. This command is used by the commands `\@@_rowcolors`, `\@@_columncolor` and `\@@_rowcolor:n` (used in `\@@_rowcolor`).

```

5908 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n \c_zero_dim }

```

Despite its name, the following command does not create a PGF path. It declares as colored by the “empty color” all the cells in what would be the path. Hence, the other coloring instructions of `nicematrix` won’t put color in those cells. the

```

5909 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5910 {
5911   \bool_set_true:N \l_@@_nocolor_used_bool
5912   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5913   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5914   \clist_map_inline:Nn \l_@@_rows_tl
5915   {
5916     \clist_map_inline:Nn \l_@@_cols_tl
5917     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5918   }
5919 }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-* and `\c@jCol` equal to 10, the clist `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

5920 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5921 {
5922   \clist_set_eq:NN \l_tmpa_clist #1

```

```

5923 \clist_clear:N #1
5924 \clist_map_inline:Nn \l_tmpa_clist
5925 {

```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```

5926 \def \l_tmpa_tl { ##1 }
5927 \tl_if_in:NnTF \l_tmpa_tl { - }
5928 { \@@_cut_on_hyphen:w ##1 \q_stop }
5929 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5930 \bool_lazy_or:nnT
5931 { \str_if_eq_p:ee \l_tmpa_tl { * } }
5932 { \tl_if_blank_p:o \l_tmpa_tl }
5933 { \def \l_tmpa_tl { 1 } }
5934 \bool_lazy_or:nnT
5935 { \str_if_eq_p:ee \l_tmpb_tl { * } }
5936 { \tl_if_blank_p:o \l_tmpb_tl }
5937 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5938 \int_compare:nNnT { \l_tmpb_tl } > { #2 }
5939 { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5940 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
5941 { \clist_put_right:Nn #1 { #####1 } }
5942 }
5943 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5944 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5945 {
5946 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5947 {

```

We must not expand the color (`#2`) because the color may contain the token `!` which may be activated by some packages (ex.: `babel` with the option `french` on `latex` and `pdflatex`).

```

5948 \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5949 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5950 }
5951 \ignorespaces
5952 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5953 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5954 {
5955 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5956 {
5957 \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5958 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5959 { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5960 }
5961 \ignorespaces
5962 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5963 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5964 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5965 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5966 {

```

A use of `\rowlistcolors` in the tabular erases the instructions `\rowlistcolors` which are in force. However, it's possible to put *several* instructions `\rowlistcolors` in the same row of a tabular: it may be useful when those instructions `\rowlistcolors` concerns different columns of the tabular (thanks to the key `cols` of `\rowlistcolors`). That's why we store the different instructions `\rowlistcolors` which are in force in a sequence `\g_@@_rowlistcolors_seq`. Now, we will filter that sequence to keep only the elements which have been issued on the actual row. We will store the elements to keep in the `\g_tmpa_seq`.

```

5967   \seq_gclear:N \g_tmpa_seq
5968   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5969     { \@@_rowlistcolors_tabular:nnnn #1 }
5970   \seq_gset_eq:NN \g_@@_rowlistcolors_seq \g_tmpa_seq

```

Now, we add to the sequence `\g_@@_rowlistcolors_seq` (which is the list of the commands `\rowlistcolors` which are in force) the current instruction `\rowlistcolors`.

```

5971   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5972   {
5973     { \int_use:N \c@iRow }
5974     { \exp_not:n { #1 } }
5975     { \exp_not:n { #2 } }
5976     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5977   }
5978   \ignorespaces
5979 }

```

The following command will be applied to each component of `\g_@@_rowlistcolors_seq`. Each component of that sequence is a kind of 4-uple of the form `{#1}{#2}{#3}{#4}`.

#1 is the number of the row where the command `\rowlistcolors` has been issued.

#2 is the colorimetric space (optional argument of the `\rowlistcolors`).

#3 is the list of colors (mandatory argument of `\rowlistcolors`).

#4 is the list of *key=value* pairs (last optional argument of `\rowlistcolors`).

```

5980 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
5981 {
5982   \int_compare:nNnTF { #1 } = { \c@iRow }

```

We (temporary) keep in memory in `\g_tmpa_seq` the instructions which will still be in force after the current instruction (because they have been issued in the same row of the tabular).

```

5983   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5984   {
5985     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5986     {
5987       \@@_rowlistcolors
5988       [ \exp_not:n { #2 } ]
5989       { #1 - \int_eval:n { \c@iRow - 1 } }
5990       { \exp_not:n { #3 } }
5991       [ \exp_not:n { #4 } ]
5992     }
5993   }
5994 }

```

The following command will be used at the end of the tabular, just before the execution of the `\g_@@_pre_code_before_tl`. It clears the sequence `\g_@@_rowlistcolors_seq` of all the commands `\rowlistcolors` which are (still) in force.

```

5995 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5996 {
5997   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5998     { \@@_rowlistcolors_tabular_ii:nnnn #1 }
5999   \seq_gclear:N \g_@@_rowlistcolors_seq
6000 }

```



```

6001 \cs_new_protected:Npn \@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
6002 {
6003   \tl_gput_right:Nn \g_@@_pre_code_before_tl
6004   { \@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
6005 }

```

The first mandatory argument of the command `\@_rowlistcolors` which is written in the `pre-\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

6006 \NewDocumentCommand \@_columncolor_preamble { 0 { } m }
6007 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```

6008   \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
6009   {

```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```

6010     \tl_gput_left:Ne \g_@@_pre_code_before_tl
6011     {
6012       \exp_not:N \columncolor [ #1 ]
6013       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
6014     }
6015   }
6016 }

```

```

6017 \cs_new_protected:Npn \@_EmptyColumn:n #1
6018 {
6019   \clist_map_inline:nn { #1 }
6020   {
6021     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6022     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6023     \columncolor { nocolor } { ##1 }
6024   }
6025 }

```

```

6026 \cs_new_protected:Npn \@_EmptyRow:n #1
6027 {
6028   \clist_map_inline:nn { #1 }
6029   {
6030     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6031     { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6032     \rowcolor { nocolor } { ##1 }
6033   }
6034 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`). That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
6035 \cs_set_eq:NN \OnlyMainNiceMatrix \use:n
```

Another definition of `\OnlyMainNiceMatrix` will be linked to the command in the environments of `nicematrix`. Here is that definition, called `\@@_OnlyMainNiceMatrix:n`.

```
6036 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6037 {
6038   \int_if_zero:nTF { \l_@@_first_col_int }
6039     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6040     {
6041       \int_if_zero:nTF { \c@jCol }
6042         {
6043           \int_compare:nNnF { \c@iRow } = { -1 }
6044             {
6045               \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
6046                 { #1 }
6047             }
6048         }
6049     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6050 }
6051 }
```

This definition may seem complicated but we must remind that the number of row `\c@iRow` is incremented in the first cell of the row, *after* a potential vertical rule on the left side of the first cell. The command `\@@_OnlyMainNiceMatrix_i:n` is only a short-cut which is used twice in the above command. This command must *not* be protected.

```
6052 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6053 {
6054   \int_if_zero:nF { \c@iRow }
6055     {
6056       \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
6057         {
6058           \int_compare:nNnT { \c@jCol } > { \c_zero_int }
6059             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6060         }
6061     }
6062 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to `-2` or `-1` (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```
6063 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6064 {
6065   \IfPackageLoadedTF { tikz }
6066     {
6067       \IfPackageLoadedTF { booktabs }
6068         { #2 }
6069         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6070     }
6071     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6072 }

6073 \NewExpandableDocumentCommand { \@@_TopRule } { }
6074 { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }

6075 \cs_new:Npn \@@_TopRule_i:
6076 {
6077   \noalign \bgroup
6078     \peek_meaning:NTF [
6079       { \@@_TopRule_ii: }

```

```

6080         { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6081     }
6082 \NewDocumentCommand \@@_TopRule_ii: { o }
6083 {
6084     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6085     {
6086         \@@_hline:n
6087         {
6088             position = \int_eval:n { \c@iRow + 1 } ,
6089             tikz =
6090             {
6091                 line~width = #1 ,
6092                 yshift = 0.25 \arrayrulewidth ,
6093                 shorten~< = - 0.5 \arrayrulewidth
6094             } ,
6095             total~width = #1
6096         }
6097     }
6098     \skip_vertical:n { \belowrulesep + #1 }
6099     \egroup
6100 }
6101 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6102 { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
6103 \cs_new:Npn \@@_BottomRule_i:
6104 {
6105     \noalign \bgroup
6106     \peek_meaning:NTF [
6107     { \@@_BottomRule_ii: }
6108     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6109 }
6110 \NewDocumentCommand \@@_BottomRule_ii: { o }
6111 {
6112     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6113     {
6114         \@@_hline:n
6115         {
6116             position = \int_eval:n { \c@iRow + 1 } ,
6117             tikz =
6118             {
6119                 line~width = #1 ,
6120                 yshift = 0.25 \arrayrulewidth ,
6121                 shorten~< = - 0.5 \arrayrulewidth
6122             } ,
6123             total~width = #1 ,
6124         }
6125     }
6126     \skip_vertical:N \aboverulesep
6127     \@@_create_row_node_i:
6128     \skip_vertical:n { #1 }
6129     \egroup
6130 }
6131 \NewExpandableDocumentCommand { \@@_MidRule } { }
6132 { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
6133 \cs_new:Npn \@@_MidRule_i:
6134 {
6135     \noalign \bgroup
6136     \peek_meaning:NTF [
6137     { \@@_MidRule_ii: }
6138     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6139 }
6140 \NewDocumentCommand \@@_MidRule_ii: { o }

```

```

6141 {
6142   \skip_vertical:N \aboverulesep
6143   \@@_create_row_node_i:
6144   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6145   {
6146     \@@_hline:n
6147     {
6148       position = \int_eval:n { \c@iRow + 1 } ,
6149       tikz =
6150       {
6151         line-width = #1 ,
6152         yshift = 0.25 \arrayrulewidth ,
6153         shorten-< = - 0.5 \arrayrulewidth
6154       } ,
6155       total-width = #1 ,
6156     }
6157   }
6158   \skip_vertical:n { \belowrulesep + #1 }
6159   \egroup
6160 }

```

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```

6161 \keys_define:nn { nicematrix / Rules }
6162 {
6163   position .int_set:N = \l_@@_position_int ,
6164   position .value_required:n = true ,
6165   start .int_set:N = \l_@@_start_int ,
6166   end .code:n =
6167     \bool_lazy_or:nnTF
6168     { \tl_if_empty_p:n { #1 } }
6169     { \str_if_eq_p:ee { #1 } { last } }
6170     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6171     { \int_set:Nn \l_@@_end_int { #1 } }
6172 }

```

It’s possible that the rule won’t be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6173 \keys_define:nn { nicematrix / RulesBis }
6174 {
6175   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6176   multiplicity .initial:n = 1 ,
6177   dotted .bool_set:N = \l_@@_dotted_bool ,
6178   dotted .initial:n = false ,
6179   dotted .default:n = true ,

```

We want that, even when the rule has been defined with TikZ by the key `tikz`, the user has still the possibility to change the color of the rule with the key `color` (in the command `\Hline`, not in the key `tikz` of the command `\Hline`). The main use is, when the user has defined its own command `\MyDashedLine` by `\newcommand{\MyDashedRule}{\Hline[tikz=dashed]}`, to give the ability to write `\MyDashedRule[color=red]`.

```

6180   color .code:n =
6181     \@@_set_CTarc:n { #1 }

```

```

6182     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6183     color .value_required:n = true ,
6184     sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
6185     sep-color .value_required:n = true ,

```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with Tikz.

```

6186     tikz .code:n =
6187         \IfPackageLoadedTF { tikz }
6188         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6189         { \@@_error:n { tikz-without-tikz } } ,
6190     tikz .value_required:n = true ,
6191     total-width .dim_set:N = \l_@@_rule_width_dim ,
6192     total-width .value_required:n = true ,
6193     width .meta:n = { total-width = #1 } ,
6194     unknown .code:n = \@@_error:n { Unknown~key~for~RulesBis }
6195 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6196 \cs_new_protected:Npn \@@_vline:n #1
6197 {

```

The group is for the options.

```

6198     \group_begin:
6199     \int_set_eq:NN \l_@@_end_int \c@iRow
6200     \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

6201     \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
6202     \@@_vline_i:
6203     \group_end:
6204 }

6205 \cs_new_protected:Npn \@@_vline_i:
6206 {

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

6207     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6208     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6209     \l_tmpa_tl
6210     {

```

The boolean `\g_tmpa_bool` indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small vertical rule won't be drawn.

```

6211     \bool_gset_true:N \g_tmpa_bool
6212     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6213     { \@@_test_vline_in_block:nnnn ##1 }
6214     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6215     { \@@_test_vline_in_block:nnnn ##1 }
6216     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6217     { \@@_test_vline_in_stroken_block:nnnn ##1 }
6218     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
6219     \bool_if:NTF \g_tmpa_bool
6220     {
6221     \int_if_zero:nT { \l_@@_local_start_int }

```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```

6222         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6223     }
6224     {
6225         \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6226         {
6227             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6228             \@@_vline_ii:
6229             \int_zero:N \l_@@_local_start_int
6230         }
6231     }
6232 }
6233 \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6234 {
6235     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6236     \@@_vline_ii:
6237 }
6238 }

6239 \cs_new_protected:Npn \@@_test_in_corner_v:
6240 {
6241     \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
6242     {
6243         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6244         { \bool_set_false:N \g_tmpa_bool }
6245     }
6246     {
6247         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6248         {
6249             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
6250             { \bool_set_false:N \g_tmpa_bool }
6251             {
6252                 \@@_if_in_corner:nT
6253                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6254                 { \bool_set_false:N \g_tmpa_bool }
6255             }
6256         }
6257     }
6258 }

6259 \cs_new_protected:Npn \@@_vline_ii:
6260 {
6261     \tl_clear:N \l_@@_tikz_rule_tl
6262     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6263     \bool_if:NTF \l_@@_dotted_bool
6264     { \@@_vline_iv: }
6265     {
6266         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6267         { \@@_vline_iii: }
6268         { \@@_vline_v: }
6269     }
6270 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6271 \cs_new_protected:Npn \@@_vline_iii:
6272 {
6273     \pgfpicture
6274     \pgfrememberpicturepositiononpagetrue
6275     \pgf@relevantforpicturesizefalse
6276     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }

```

```

6277 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6278 \@_qpoint:n { col - \int_use:N \l_@@_position_int }
6279 \dim_set:Nn \l_tmpb_dim
6280 {
6281   \pgf@x
6282   - 0.5 \l_@@_rule_width_dim
6283   +
6284   ( \arrayrulewidth * \l_@@_multiplicity_int
6285     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6286 }
6287 \@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6288 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6289 \bool_lazy_all:nT
6290 {
6291   { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6292   { \cs_if_exist_p:N \CT@drsc@ }
6293   { ! \tl_if_blank_p:o \CT@drsc@ }
6294 }
6295 {
6296   \group_begin:
6297   \CT@drsc@
6298   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6299   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6300   \dim_set:Nn \l_@@_tmpd_dim
6301   {
6302     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6303     * ( \l_@@_multiplicity_int - 1 )
6304   }
6305   \pgfpathrectanglecorners
6306   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6307   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6308   \pgfusepath { fill }
6309   \group_end:
6310 }
6311 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6312 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6313 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6314 {
6315   \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6316   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6317   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6318 }
6319 \CT@arc@
6320 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6321 \pgfsetrectcap
6322 \pgfusepathqstroke
6323 \endpgfpicture
6324 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6325 \cs_new_protected:Npn \@_vline_iv:
6326 {
6327   \pgfpicture
6328   \pgfrememberpicturepositiononpagetrue
6329   \pgf@relevantforpicturesizefalse
6330   \@_qpoint:n { col - \int_use:N \l_@@_position_int }
6331   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6332   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6333   \@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6334   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6335   \@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6336   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6337   \CT@arc@

```

```

6338 \@@_draw_line:
6339 \endpgfpicture
6340 }

```

The following code is for the case when the user uses the key `tikz`.

```

6341 \cs_new_protected:Npn \@@_vline_v:
6342 {
6343 \begin { tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6344 \CT@arc@
6345 \tl_if_empty:NF \l_@@_rule_color_tl
6346 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6347 \pgfrememberpicturepositiononpagetrue
6348 \pgf@relevantforpicturesizefalse
6349 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6350 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6351 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6352 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6353 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6354 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6355 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6356 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6357 ( \l_tmpb_dim , \l_tmpa_dim ) --
6358 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6359 \end { tikzpicture }
6360 }

```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```

6361 \cs_new_protected:Npn \@@_draw_vlines:
6362 {
6363 \int_step_inline:nnn
6364 {
6365 \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6366 { 2 }
6367 { 1 }
6368 }
6369 {
6370 \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6371 { \c@jCol }
6372 { \int_eval:n { \c@jCol + 1 } }
6373 }
6374 {
6375 \str_if_eq:eeF \l_@@_vlines_clist { all }
6376 { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6377 { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6378 }
6379 }

```

The horizontal rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs of the form `{nicematrix/Rules}`.

```

6380 \cs_new_protected:Npn \@@_hline:n #1
6381 {

```


The group is for the options.

```

6382   \group_begin:
6383   \int_set_eq:NN \l_@@_end_int \c@jCol
6384   \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6385   \@@_hline_i:
6386   \group_end:
6387 }

6388 \cs_new_protected:Npn \@@_hline_i:
6389 {
6390   % \int_zero:N \l_@@_local_start_int
6391   % \int_zero:N \l_@@_local_end_int

```

$\l_1\text{tmpa_tl}$ is the number of row and $\l_1\text{tmpb_tl}$ the number of column. When we have found a column corresponding to a rule to draw, we note its number in $\l_1\text{@@_tmpc_tl}$.

```

6392   \tl_set:No \l_1tmpa_tl { \int_use:N \l_@@_position_int }
6393   \int_step_variable:nNn \l_@@_start_int \l_@@_end_int
6394     \l_1tmpb_tl
6395   {

```

The boolean $\g_1\text{tmpa_bool}$ indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by $\backslash\text{Block}$ or a virtual block corresponding to a dotted line, created by $\backslash\text{Cdots}$, $\backslash\text{Vdots}$, etc.), we will set $\g_1\text{tmpa_bool}$ to false and the small horizontal rule won't be drawn.

```

6396     \bool_gset_true:N \g_1tmpa_bool

```

We test whether we are in a block.

```

6397     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6398       { \@@_test_hline_in_block:nnnnn ##1 }

6399     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6400       { \@@_test_hline_in_block:nnnnn ##1 }
6401     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6402       { \@@_test_hline_in_stroken_block:nnnn ##1 }
6403     \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
6404     \bool_if:NTF \g_1tmpa_bool
6405       {
6406         \int_if_zero:nT { \l_@@_local_start_int }

```

We keep in memory that we have a rule to draw. $\l_1\text{@@_local_start_int}$ will be the starting row of the rule that we will have to draw.

```

6407         { \int_set:Nn \l_@@_local_start_int \l_1tmpb_tl }
6408       }
6409     {
6410       \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6411       {
6412         \int_set:Nn \l_@@_local_end_int { \l_1tmpb_tl - 1 }
6413         \@@_hline_ii:
6414         \int_zero:N \l_@@_local_start_int
6415       }
6416     }
6417   }
6418   \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
6419   {
6420     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6421     \@@_hline_ii:
6422   }
6423 }

```

```

6424 \cs_new_protected:Npn \@@_test_in_corner_h:
6425 {
6426   \int_compare:nNnTF { \l_1tmpa_tl } = { \c@iRow + 1 }
6427   {
6428     \@@_if_in_corner:nT { \int_eval:n { \l_1tmpa_tl - 1 } - \l_1tmpb_tl }

```

```

6429         { \bool_set_false:N \g_tmpa_bool }
6430     }
6431     {
6432     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6433     {
6434         \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
6435         { \bool_set_false:N \g_tmpa_bool }
6436         {
6437             \@@_if_in_corner:nT
6438             { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6439             { \bool_set_false:N \g_tmpa_bool }
6440         }
6441     }
6442 }
6443 }

```

```

6444 \cs_new_protected:Npn \@@_hline_ii:
6445 {
6446     \tl_clear:N \l_@@_tikz_rule_tl
6447     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6448     \bool_if:NTF \l_@@_dotted_bool
6449     { \@@_hline_iv: }
6450     {
6451         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6452         { \@@_hline_iii: }
6453         { \@@_hline_v: }
6454     }
6455 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6456 \cs_new_protected:Npn \@@_hline_iii:
6457 {
6458     \pgfpicture
6459     \pgfrememberpicturepositiononpagetrue
6460     \pgf@relevantforpicturesizefalse
6461     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6462     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6463     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6464     \dim_set:Nn \l_tmpb_dim
6465     {
6466         \pgf@y
6467         - 0.5 \l_@@_rule_width_dim
6468         +
6469         ( \arrayrulewidth * \l_@@_multiplicity_int
6470         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6471     }
6472     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6473     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6474     \bool_lazy_all:nT
6475     {
6476         { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
6477         { \cs_if_exist_p:N \CT@drsc@ }
6478         { ! \tl_if_blank_p:o \CT@drsc@ }
6479     }
6480     {
6481         \group_begin:
6482         \CT@drsc@
6483         \dim_set:Nn \l_@@_tmpd_dim
6484         {
6485             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6486             * ( \l_@@_multiplicity_int - 1 )
6487         }

```

```

6488     \pgfpathrectanglecorners
6489     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6490     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6491     \pgfusepathqfill
6492     \group_end:
6493 }
6494 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6495 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6496 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6497 {
6498     \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
6499     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6500     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6501 }
6502 \CT@arc@
6503 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6504 \pgfsetrectcap
6505 \pgfusepathqstroke
6506 \endpgfpicture
6507 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6508 \cs_new_protected:Npn \l_@@_hline_iv:
6509 {
6510     \pgfpicture
6511     \pgfrememberpicturepositiononpagetrue
6512     \pgf@relevantforpicturesizefalse
6513     \l_@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6514     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6515     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6516     \l_@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6517     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6518     \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
6519     {
6520         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6521         \bool_if:NF \g_@@_delims_bool
6522         { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_xdots_inter_dim` is *ad hoc* for a better result.

```

6523     \tl_if_eq:NnF \g_@@_left_delim_tl (
6524     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6525     )
6526     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6527     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6528     \int_compare:nNnT { \l_@@_local_end_int } = { \c_jCol }

```

```

6529   {
6530     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6531     \bool_if:NF \g_@@_delims_bool
6532     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6533     \tl_if_eq:NnF \g_@@_right_delim_tl )
6534     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6535   }
6536   \CT@arc@
6537   \@@_draw_line:
6538   \endpgfpicture
6539 }

```

The following code is for the case when the user uses the key `tikz` (in the definition of a customized rule by using the key `custom-line`).

```

6540 \cs_new_protected:Npn \@@_hline_v:
6541   {
6542     \begin { tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6543     \CT@arc@
6544     \tl_if_empty:NF \l_@@_rule_color_tl
6545     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6546     \pgfrememberpicturepositiononpagetrue
6547     \pgf@relevantforpicturesizefalse
6548     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6549     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6550     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6551     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6552     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6553     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6554     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6555     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6556     ( \l_tmpa_dim , \l_tmpb_dim ) --
6557     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6558     \end { tikzpicture }
6559 }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners — if the key `corners` is used).

```

6560 \cs_new_protected:Npn \@@_draw_hlines:
6561   {
6562     \int_step_inline:nnn
6563     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6564     {
6565       \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
6566       { \c@iRow }
6567       { \int_eval:n { \c@iRow + 1 } }
6568     }
6569     {
6570       \str_if_eq:eeF \l_@@_hlines_clist { all }
6571       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6572       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6573     }
6574   }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6575 \cs_set:Npn \@@_Hline: { \noalign \bgroup \@@_Hline_i:n { 1 } }

```

The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

6576 \cs_set:Npn \@@_Hline_i:n #1
6577   {
6578     \peek_remove_spaces:n
6579     {
6580       \peek_meaning:NTF \Hline
6581       { \@@_Hline_ii:nn { #1 + 1 } }
6582       { \@@_Hline_iii:n { #1 } }
6583     }
6584   }
6585 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6586 \cs_set:Npn \@@_Hline_iii:n #1
6587   { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6588 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6589   {
6590     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6591     \skip_vertical:N \l_@@_rule_width_dim
6592     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6593     {
6594       \@@_hline:n
6595       {
6596         multiplicity = #1 ,
6597         position = \int_eval:n { \c@iRow + 1 } ,
6598         total-width = \dim_use:N \l_@@_rule_width_dim ,
6599         #2
6600       }
6601     }
6602   \egroup
6603 }

```

Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of `key=value` pairs.

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```

6604 \cs_new_protected:Npn \@@_custom_line:n #1
6605   {
6606     \str_clear_new:N \l_@@_command_str
6607     \str_clear_new:N \l_@@_ccommand_str
6608     \str_clear_new:N \l_@@_letter_str
6609     \tl_clear_new:N \l_@@_other_keys_tl
6610     \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6611   \bool_lazy_all:nTF
6612     {
6613       { \str_if_empty_p:N \l_@@_letter_str }
6614       { \str_if_empty_p:N \l_@@_command_str }
6615       { \str_if_empty_p:N \l_@@_ccommand_str }
6616     }
6617     { \@@_error:n { No~letter~and~no~command } }
6618     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6619   }
6620 \keys_define:nn { nicematrix / custom-line }
6621   {
6622     letter .str_set:N = \l_@@_letter_str ,

```

```

6623 letter .value_required:n = true ,
6624 command .str_set:N = \l_@@_command_str ,
6625 command .value_required:n = true ,
6626 ccommand .str_set:N = \l_@@_ccommand_str ,
6627 ccommand .value_required:n = true ,
6628 }

```

```

6629 \cs_new_protected:Npn \@@_custom_line_i:n #1
6630 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6631 \bool_set_false:N \l_@@_tikz_rule_bool
6632 \bool_set_false:N \l_@@_dotted_rule_bool
6633 \bool_set_false:N \l_@@_color_bool
6634 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6635 \bool_if:NT \l_@@_tikz_rule_bool
6636 {
6637   \IfPackageLoadedF { tikz }
6638   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6639   \bool_if:NT \l_@@_color_bool
6640   { \@@_error:n { color-in-custom-line-with-tikz } }
6641 }
6642 \bool_if:NT \l_@@_dotted_rule_bool
6643 {
6644   \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
6645   { \@@_error:n { key-multiplicity-with-dotted } }
6646 }
6647 \str_if_empty:NF \l_@@_letter_str
6648 {
6649   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6650   { \@@_error:n { Several-letters } }
6651   {
6652     \tl_if_in:NoTF
6653     \c_@@_forbidden_letters_str
6654     \l_@@_letter_str
6655     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6656   }

```

During the analyse of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6657 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
6658 { \@@_v_custom_line:n { #1 } }
6659 }
6660 }
6661 }
6662 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6663 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6664 }
6665 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6666 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6667 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6668 \keys_define:nn { nicematrix / custom-line-bis }
6669 {
6670   multiplicity .int_set:N = \l_@@_multiplicity_int ,

```

```

6671     multiplicity .initial:n = 1 ,
6672     multiplicity .value_required:n = true ,
6673     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6674     color .value_required:n = true ,
6675     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6676     tikz .value_required:n = true ,
6677     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6678     dotted .value_forbidden:n = true ,
6679     total-width .code:n = { } ,
6680     total-width .value_required:n = true ,
6681     width .code:n = { } ,
6682     width .value_required:n = true ,
6683     sep-color .code:n = { } ,
6684     sep-color .value_required:n = true ,
6685     unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6686 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6687 \bool_new:N \l_@@_dotted_rule_bool
6688 \bool_new:N \l_@@_tikz_rule_bool
6689 \bool_new:N \l_@@_color_bool

```

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key `width` is deprecated and has been replaced by the key `total-width`.

```

6690 \keys_define:nn { nicematrix / custom-line-width }
6691 {
6692     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6693     multiplicity .initial:n = 1 ,
6694     multiplicity .value_required:n = true ,
6695     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6696     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6697         \bool_set_true:N \l_@@_total_width_bool ,
6698     total-width .value_required:n = true ,
6699     width .meta:n = { total-width = #1 } ,
6700     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6701 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6702 \cs_new_protected:Npn \@@_h_custom_line:n #1
6703 {

```

We use `\cs_set:cpn` and not `\cs_new:cpn` because we want a local definition. Moreover, the command must *not* be protected since it begins with `\noalign` (which is in `\Hline`).

```

6704     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6705     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6706 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter `c` as in `\cline`). `#1` is the whole set of keys to pass to the command `\@@_hline:n` (which is in the internal `\CodeAfter`).

```

6707 \cs_new_protected:Npn \@@_c_custom_line:n #1
6708 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6709     \exp_args:Nc \NewExpandableDocumentCommand
6710     { nicematrix - \l_@@_ccommand_str }
6711     { 0 { } m }

```

```

6712 {
6713   \noalign
6714   {
6715     \@@_compute_rule_width:n { #1 , ##1 }
6716     \skip_vertical:n { \l_@@_rule_width_dim }
6717     \clist_map_inline:nn
6718       { ##2 }
6719       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6720   }
6721 }
6722 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6723 }

```

The first argument is the list of key-value pairs characteristic of the line. The second argument is the specification of columns for the `\cline` with the syntax *a-b*.

```

6724 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6725 {
6726   \tl_if_in:nnTF { #2 } { - }
6727     { \@@_cut_on_hyphen:w #2 \q_stop }
6728     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6729   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6730   {
6731     \@@_hline:n
6732     {
6733       #1 ,
6734       start = \l_tmpa_tl ,
6735       end = \l_tmpb_tl ,
6736       position = \int_eval:n { \c@iRow + 1 } ,
6737       total-width = \dim_use:N \l_@@_rule_width_dim
6738     }
6739   }
6740 }
6741 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6742 {
6743   \bool_set_false:N \l_@@_tikz_rule_bool
6744   \bool_set_false:N \l_@@_total_width_bool
6745   \bool_set_false:N \l_@@_dotted_rule_bool
6746   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6747   \bool_if:NF \l_@@_total_width_bool
6748   {
6749     \bool_if:NTF \l_@@_dotted_rule_bool
6750       { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6751       {
6752         \bool_if:NF \l_@@_tikz_rule_bool
6753         {
6754           \dim_set:Nn \l_@@_rule_width_dim
6755             {
6756               \arrayrulewidth * \l_@@_multiplicity_int
6757               + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6758             }
6759         }
6760       }
6761   }
6762 }
6763 \cs_new_protected:Npn \@@_v_custom_line:n #1
6764 {
6765   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6766   \tl_gput_right:Ne \g_@@_array_preamble_tl
6767     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6768   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6769   {

```



```

6770     \@@_vline:n
6771     {
6772         #1 ,
6773         position = \int_eval:n { \c@jCol + 1 } ,
6774         total-width = \dim_use:N \l_@@_rule_width_dim
6775     }
6776 }
6777 \@@_rec_preamble:n
6778 }
6779 \@@_custom_line:n
6780 { letter = : , command = hdottedline , ccommand = cdottedline, dotted }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```

6781 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6782 {
6783     \int_compare:nNnT { \l_tmpa_tl } > { #1 }
6784     {
6785         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6786         {
6787             \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6788             {
6789                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6790                 { \bool_gset_false:N \g_tmpa_bool }
6791             }
6792         }
6793     }
6794 }

```

The same for vertical rules.

```

6795 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6796 {
6797     \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6798     {
6799         \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6800         {
6801             \int_compare:nNnT { \l_tmpb_tl } > { #2 }
6802             {
6803                 \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6804                 { \bool_gset_false:N \g_tmpa_bool }
6805             }
6806         }
6807     }
6808 }
6809 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6810 {
6811     \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
6812     {
6813         \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
6814         {
6815             \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
6816             { \bool_gset_false:N \g_tmpa_bool }
6817             {
6818                 \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
6819                 { \bool_gset_false:N \g_tmpa_bool }
6820             }
6821         }
6822     }
6823 }

```

```

6824 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6825 {
6826   \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
6827   {
6828     \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
6829     {
6830       \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
6831       { \bool_gset_false:N \g_tmpa_bool }
6832       {
6833         \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
6834         { \bool_gset_false:N \g_tmpa_bool }
6835       }
6836     }
6837   }
6838 }

```

23 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```

6839 \cs_new_protected:Npn \@@_compute_corners:
6840 {
6841   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6842   { \@@_mark_cells_of_block:nnnnn ##1 }

```

The list `\l_@@_corners_cells_clist` will be the list of all the empty cells (and not in a block) considered in the corners of the array. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

6843   \clist_clear:N \l_@@_corners_cells_clist
6844   \clist_map_inline:Nn \l_@@_corners_clist
6845   {
6846     \str_case:nnF { ##1 }
6847     {
6848       { NW }
6849       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6850       { NE }
6851       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6852       { SW }
6853       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6854       { SE }
6855       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6856     }
6857     { \@@_error:nn { bad~corner } { ##1 } }
6858   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6859   \clist_if_empty:NF \l_@@_corners_cells_clist
6860   {

```

You write on the aux file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which colors the rows, columns and cells must not color the cells in the corners.

```

6861     \tl_gput_right:Ne \g_@@_aux_tl
6862     {
6863       \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
6864       { \l_@@_corners_cells_clist }
6865     }
6866   }
6867 }

```

```

6868 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6869 {
6870   \int_step_inline:nnn { #1 } { #3 }
6871   {
6872     \int_step_inline:nnn { #2 } { #4 }
6873     { \cs_set_nopar:cpn { @@_block_ ##1 - ###1 } { } }
6874   }
6875 }

6876 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6877 {
6878   \cs_if_exist:cTF
6879   { @@_block_ \int_eval:n { #1 } - \int_eval:n { #2 } }
6880   { \prg_return_true: }
6881   { \prg_return_false: }
6882 }

```

“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence `\l_@@_corners_cells_clist`.

The six arguments of `\@@_compute_a_corner:nnnnnn` are as follow:

- #1 and #2 are the number of row and column of the cell which is actually in the corner;
- #3 and #4 are the steps in rows and the step in columns when moving from the corner;
- #5 is the number of the final row when scanning the rows from the corner;
- #6 is the number of the final column when scanning the columns from the corner.

```

6883 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6884 {

```

For the explanations and the name of the variables, we consider that we are computing the left-upper corner.

First, we try to determine which is the last empty cell (and not in a block: we won't add that precision any longer) in the column of number 1. The flag `\l_tmpa_bool` will be raised when a non-empty cell is found.

```

6885   \bool_set_false:N \l_tmpa_bool
6886   \int_zero_new:N \l_@@_last_empty_row_int
6887   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6888   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6889   {
6890     \bool_lazy_or:nnTF
6891     {
6892       \cs_if_exist_p:c
6893       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6894     }
6895     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6896     { \bool_set_true:N \l_tmpa_bool }
6897   }
6898   \bool_if:NF \l_tmpa_bool
6899   { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6900 }
6901 }

```

Now, you determine the last empty cell in the row of number 1.

```

6902   \bool_set_false:N \l_tmpa_bool
6903   \int_zero_new:N \l_@@_last_empty_column_int
6904   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6905   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6906   {
6907     \bool_lazy_or:nnTF
6908     {

```

```

6909     \cs_if_exist_p:c
6910     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6911   }
6912   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6913   { \bool_set_true:N \l_tmpa_bool }
6914   {
6915     \bool_if:NF \l_tmpa_bool
6916     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6917   }
6918 }

```

Now, we loop over the rows.

```

6919   \int_step_inline:nnnn { #1 } { #3 } { \l_@@_last_empty_row_int }
6920   {

```

We treat the row number `##1` with another loop.

```

6921     \bool_set_false:N \l_tmpa_bool
6922     \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
6923     {
6924       \bool_lazy_or:nnTF
6925       { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6926       { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6927       { \bool_set_true:N \l_tmpa_bool }
6928     {
6929       \bool_if:NF \l_tmpa_bool
6930       {
6931         \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6932         \clist_put_right:Nn
6933         \l_@@_corners_cells_clist
6934         { ##1 - #####1 }
6935         \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6936       }
6937     }
6938   }
6939 }
6940 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6941 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6942 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...`

24 The environment `{NiceMatrixBlock}`

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

6943 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6944 \keys_define:nn { nicematrix / NiceMatrixBlock }
6945 {
6946   auto-columns-width .code:n =
6947   {
6948     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6949     \dim_gzero_new:N \g_@@_max_cell_width_dim
6950     \bool_set_true:N \l_@@_auto_columns_width_bool
6951   }
6952 }

```

```

6953 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6954 {
6955   \int_gincr:N \g_@@_NiceMatrixBlock_int
6956   \dim_zero:N \l_@@_columns_width_dim
6957   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6958   \bool_if:NT \l_@@_block_auto_columns_width_bool
6959     {
6960       \cs_if_exist:cT
6961         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6962         {
6963           \dim_set:Nn \l_@@_columns_width_dim
6964             {
6965               \use:c
6966                 { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6967             }
6968         }
6969     }
6970 }

```

At the end of the environment `{NiceMatrixBlock}`, we write in the main aux file instructions for the column width of all the environments of the block (that's why we have stored the number of the first environment of the block in the counter `\l_@@_first_env_block_int`).

```

6971 {
6972   \legacy_if:nTF { measuring@ }

```

If `{NiceMatrixBlock}` is used in an environment of `amsmath` such as `{align}`: cf. question 694957 on TeX StackExchange. The most important line in that case is the following one.

```

6973   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6974   {
6975     \bool_if:NT \l_@@_block_auto_columns_width_bool
6976     {
6977       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6978       \iow_shipout:Ne \@mainaux
6979         {
6980           \cs_gset:cpn
6981             { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }

```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```

6982         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6983       }
6984       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6985     }
6986   }
6987   \ignorespacesafterend
6988 }

```

25 The extra nodes

The following command is called in `\@@_use_arraybox_with_notes_c`: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

```

6989 \cs_new_protected:Npn \@@_create_extra_nodes:
6990 {
6991   \bool_if:nTF \l_@@_medium_nodes_bool
6992     {
6993       \bool_if:NTF \l_@@_no_cell_nodes_bool
6994         { \@@_error:n { extra-nodes~with~no~cell~nodes } }
6995       {
6996         \bool_if:NTF \l_@@_large_nodes_bool

```

```

6997         \@@_create_medium_and_large_nodes:
6998         \@@_create_medium_nodes:
6999     }
7000 }
7001 {
7002     \bool_if:NT \l_@@_large_nodes_bool
7003     {
7004         \bool_if:NTF \l_@@_no_cell_nodes_bool
7005         { \@@_error:n { extra-nodes-with-no-cell-nodes } }
7006         \@@_create_large_nodes:
7007     }
7008 }
7009 }

```

We have three macros of creation of nodes: `\@@_create_medium_nodes:`, `\@@_create_large_nodes:` and `\@@_create_medium_and_large_nodes:`.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command `\@@_computations_for_medium_nodes:` to do these computations.

The command `\@@_computations_for_medium_nodes:` must be used in a `{pgfpicture}`.

For each row i , we compute two dimensions `l_@@_row_i_min_dim` and `l_@@_row_i_max_dim`. The dimension `l_@@_row_i_min_dim` is the minimal y -value of all the cells of the row i . The dimension `l_@@_row_i_max_dim` is the maximal y -value of all the cells of the row i .

Similarly, for each column j , we compute two dimensions `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. The dimension `l_@@_column_j_min_dim` is the minimal x -value of all the cells of the column j . The dimension `l_@@_column_j_max_dim` is the maximal x -value of all the cells of the column j .

Since these dimensions will be computed as maximum or minimum, we initialize them to `\c_max_dim` or `-\c_max_dim`.

```

7010 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7011 {
7012     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7013     {
7014         \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
7015         \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
7016         \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
7017         \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
7018     }
7019     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7020     {
7021         \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
7022         \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
7023         \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
7024         \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
7025     }

```

We begin the two nested loops over the rows and the columns of the array.

```

7026     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7027     {
7028         \int_step_variable:nnNn
7029         \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

If the cell $(i-j)$ is empty or an implicit cell (that is to say a cell after implicit ampersands `&`) we don’t update the dimensions we want to compute.

```

7030     {
7031         \cs_if_exist:cT
7032         { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell ($i-j$). They will be stored in `\pgf@x` and `\pgf@y`.

```

7033     {
7034         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7035         \dim_set:cn { l_@@_row _ \@@_i: _min_dim }
7036         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7037         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7038         {
7039             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7040             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7041         }

```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell ($i-j$). They will be stored in `\pgf@x` and `\pgf@y`.

```

7042         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7043         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7044         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } { \pgf@y } }
7045         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7046         {
7047             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7048             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
7049         }
7050     }
7051 }
7052 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7053 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7054 {
7055     \dim_compare:nNnT
7056     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7057     {
7058         \@@_qpoint:n { row - \@@_i: - base }
7059         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7060         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7061     }
7062 }
7063 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7064 {
7065     \dim_compare:nNnT
7066     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7067     {
7068         \@@_qpoint:n { col - \@@_j: }
7069         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7070         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7071     }
7072 }
7073 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7074 \cs_new_protected:Npn \@@_create_medium_nodes:
7075 {
7076     \pgfpicture
7077     \pgfrememberpicturepositiononpagetrue
7078     \pgf@relevantforpicturesizefalse
7079     \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

7080     \tl_set:Nn \l_@@_suffix_tl { -medium }
7081     \@@_create_nodes:

```

```

7082   \endpgfpicture
7083 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁵. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7084 \cs_new_protected:Npn \@@_create_large_nodes:
7085 {
7086   \pgfpicture
7087   \pgfrememberpicturepositiononpagetrue
7088   \pgf@relevantforpicturesizefalse
7089   \@@_computations_for_medium_nodes:
7090   \@@_computations_for_large_nodes:
7091   \tl_set:Nn \l_@@_suffix_tl { - large }
7092   \@@_create_nodes:
7093   \endpgfpicture
7094 }

7095 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7096 {
7097   \pgfpicture
7098   \pgfrememberpicturepositiononpagetrue
7099   \pgf@relevantforpicturesizefalse
7100   \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

7101   \tl_set:Nn \l_@@_suffix_tl { - medium }
7102   \@@_create_nodes:
7103   \@@_computations_for_large_nodes:
7104   \tl_set:Nn \l_@@_suffix_tl { - large }
7105   \@@_create_nodes:
7106   \endpgfpicture
7107 }

```

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

7108 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7109 {
7110   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7111   \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

7112   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7113   {
7114     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7115     {
7116       (
7117         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7118         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7119       )
7120       / 2
7121     }
7122     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7123     { l_@@_row _ \@@_i: _ min _ dim }
7124   }
7125   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:

```

¹⁵If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`


```

7126 {
7127   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7128   {
7129     (
7130       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7131       \dim_use:c
7132         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7133     )
7134     / 2
7135   }
7136   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7137   { l_@@_column _ \@@_j: _ max _ dim }
7138 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7139   \dim_sub:cn
7140     { l_@@_column _ 1 _ min _ dim }
7141   \l_@@_left_margin_dim
7142   \dim_add:cn
7143     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7144   \l_@@_right_margin_dim
7145 }

```

The command `\@@_create_nodes:` is used twice: for the construction of the “medium nodes” and for the construction of the “large nodes”. The nodes are constructed with the value of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. Between the construction of the “medium nodes” and the “large nodes”, the values of these dimensions are changed.

The function also uses `\l_@@_suffix_tl` (-medium or -large).

```

7146 \cs_new_protected:Npn \@@_create_nodes:
7147 {
7148   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7149   {
7150     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7151     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7152       \@@_pgf_rect_node:nnnnn
7153       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7154       { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7155       { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7156       { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7157       { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7158       \str_if_empty:NF \l_@@_name_str
7159       {
7160         \pgfnodealias
7161           { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7162           { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7163       }
7164     }
7165   }
7166   \int_step_inline:nn { \c@iRow }
7167   {
7168     \pgfnodealias
7169       { \@@_env: - ##1 - last \l_@@_suffix_tl }
7170       { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
7171   }
7172   \int_step_inline:nn { \c@jCol }
7173   {
7174     \pgfnodealias
7175       { \@@_env: - last - ##1 \l_@@_suffix_tl }
7176       { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
7177   }

```

```

7178 \pgfnodealias % added 2025-04-05
7179 { \@@_env: - last - last \l_@@_suffix_tl }
7180 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol \l_@@_suffix_tl }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with $n > 1$ was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of n .

```

7181 \seq_map_pairwise_function:NNN
7182 \g_@@_multicolumn_cells_seq
7183 \g_@@_multicolumn_sizes_seq
7184 \@@_node_for_multicolumn:nn
7185 }

```

```

7186 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7187 {
7188   \cs_set_nopar:Npn \@@_i: { #1 }
7189   \cs_set_nopar:Npn \@@_j: { #2 }
7190 }

```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{n}{...}{...}` was issued in the format i - j and the second is the value of n (the length of the “multi-cell”).

```

7191 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7192 {
7193   \@@_extract_coords_values: #1 \q_stop
7194   \@@_pgf_rect_node:nnnnn
7195   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7196   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7197   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7198   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7199   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7200   \str_if_empty:NF \l_@@_name_str
7201   {
7202     \pgfnodealias
7203     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7204     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7205   }
7206 }

```

26 The blocks

The following code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass (in the cell of the array).

```

7207 \keys_define:nn { nicematrix / Block / FirstPass }
7208 {
7209   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7210           \bool_set_true:N \l_@@_p_block_bool ,
7211   j .value_forbidden:n = true ,
7212   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7213   l .value_forbidden:n = true ,
7214   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7215   r .value_forbidden:n = true ,
7216   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7217   c .value_forbidden:n = true ,

```

```

7218 L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7219 L .value_forbidden:n = true ,
7220 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7221 R .value_forbidden:n = true ,
7222 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7223 C .value_forbidden:n = true ,
7224 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7225 t .value_forbidden:n = true ,
7226 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7227 T .value_forbidden:n = true ,
7228 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7229 b .value_forbidden:n = true ,
7230 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7231 B .value_forbidden:n = true ,
7232 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7233 m .value_forbidden:n = true ,
7234 v-center .meta:n = m ,
7235 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7236 p .value_forbidden:n = true ,
7237 color .code:n =
7238   \@@_color:n { #1 }
7239   \tl_set_rescan:Nnn
7240     \l_@@_draw_tl
7241     { \char_set_catcode_other:N ! }
7242     { #1 } ,
7243   color .value_required:n = true ,
7244   respect-arraystretch .code:n =
7245     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7246   respect-arraystretch .value_forbidden:n = true ,
7247 }

```

The following command `\@@_Block:` will be linked to `\Block` in the environments of `nicematrix`. We define it with `\NewExpandableDocumentCommand` because it has an optional argument between `<` and `>`. It's mandatory to use an expandable command.

```

7248 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7249 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7250 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not been provided by the user, you use 1-1 (that is to say a block of only one cell).

```

7251   \tl_if_blank:nTF { #2 }
7252     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7253     {
7254       \tl_if_in:nnTF { #2 } { - }
7255       {
7256         \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7257         \@@_Block_i_czech:w \@@_Block_i:w
7258         #2 \q_stop
7259       }
7260       {
7261         \@@_error:nn { Bad~argument~for~Block } { #2 }
7262         \@@_Block_ii:nnnnn \c_one_int \c_one_int
7263       }
7264     }
7265   { #1 } { #3 } { #4 }
7266   \ignorespaces
7267 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7268 \cs_new:Npn \@@_Block_i:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

With `babel` with the key `czech`, the character `-` (hyphen) is active. That's why we need a special version. Remark that we could not use a preprocessor in the command `\@@_Block:` to do the job because the command `\@@_Block:` is defined with the command `\NewExpandableDocumentCommand`.

```

7269 {
7270   \char_set_catcode_active:N -
7271   \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7272 }

```

Now, the arguments have been extracted: `#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of `key=values` pairs, `#4` are the tokens to put before the math mode and before the composition of the block and `#5` is the label (=content) of the block.

```

7273 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7274 {

```

We recall that `#1` and `#2` have been extracted from the first mandatory argument of `\Block` (which is of the syntax $i-j$). However, the user is allowed to omit i or j (or both). We detect that situation by replacing a missing value by 100 (it's a convention: when the block will actually be drawn these values will be detected and interpreted as *maximal possible value* according to the actual size of the array).

```

7275   \bool_lazy_or:nnTF
7276     { \tl_if_blank_p:n { #1 } }
7277     { \str_if_eq_p:ee { * } { #1 } }
7278     { \int_set:Nn \l_tmpa_int { 100 } }
7279     { \int_set:Nn \l_tmpa_int { #1 } }
7280   \bool_lazy_or:nnTF
7281     { \tl_if_blank_p:n { #2 } }
7282     { \str_if_eq_p:ee { * } { #2 } }
7283     { \int_set:Nn \l_tmpb_int { 100 } }
7284     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7285   \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
7286     {
7287       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7288         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7289         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7290     }
7291     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }

```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```

7292   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7293   \tl_set:Ne \l_tmpa_tl
7294     {
7295       { \int_use:N \c@iRow }
7296       { \int_use:N \c@jCol }
7297       { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7298       { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7299     }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets: `{imin}{jmin}{imax}{jmax}`.

We have different treatments when the key `p` is used and when the block is mono-column or mono-row, etc. That's why we have several macros: `\@@_Block_iv:nnnnn`, `\@@_Block_v:nnnnn`, `\@@_Block_vi:nnnn`, etc. (the five arguments of those macros are provided by curryfication).

```

7300   \bool_set_false:N \l_tmpa_bool
7301   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7302     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7303   \bool_case:nF
7304     {
7305       \l_tmpa_bool                { \@@_Block_vii:eennn }
7306       \l_@@_p_block_bool          { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7307     \l_@@_X_bool                { \@@_Block_v:eennn }
7308     { \tl_if_empty_p:n { #5 } }  { \@@_Block_v:eennn }
7309     { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7310     { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7311   }
7312   { \@@_Block_v:eennn }
7313   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7314 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don't use the key p. In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn` which will do the main job.

#1 is *i* (the number of rows of the block), #2 is *j* (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the potential math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7315 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7316 {
7317   \int_gincr:N \g_@@_block_box_int
7318   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7319     {
7320       \tl_gput_right:Ne \g_@@_pre_code_after_tl
7321         {
7322           \@@_actually_diagbox:nnnnn
7323             { \int_use:N \c@iRow }
7324             { \int_use:N \c@jCol }
7325             { \int_eval:n { \c@iRow + #1 - 1 } }
7326             { \int_eval:n { \c@jCol + #2 - 1 } }
7327             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7328             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7329         }
7330     }
7331   \box_gclear_new:c
7332     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7333   \hbox_gset:cn
7334     { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }
7335   {

```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with `\set@color` and not `\color_ensure_current:` (in order to use `\color_ensure_current:` safely, you should load `l3backend` before the `\documentclass`).

```

7336     \tl_if_empty:NTF \l_@@_color_tl

```

```

7337         { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
7338         { \@@_color:o \l_@@_color_tl }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

7339         \int_compare:nNnT { #1 } = { \c_one_int }
7340         {
7341             \int_if_zero:nTF { \c_iRow }
7342             {

```

In the following code, the value of `code-for-first-row` contains a `\Block` (in order to have the “first row” centered). But, that block will be executed, since it is entirely contained in the first row, the value of `code-for-first-row` will be inserted once again... with the same command `\Block`. That’s why we have to nullify the command `\Block`.

```

 $\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$ 

```

```

7343         \cs_set_eq:NN \Block \@@_NullBlock:
7344         \l_@@_code_for_first_row_tl
7345         }
7346         {
7347             \int_compare:nNnT { \c_iRow } = { \l_@@_last_row_int }
7348             {
7349                 \cs_set_eq:NN \Block \@@_NullBlock:
7350                 \l_@@_code_for_last_row_tl
7351             }
7352         }
7353         \g_@@_row_style_tl
7354     }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7355         \@@_reset_arraystretch:
7356         \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7357         #4

```

We adjust `\l_@@_hpos_block_str` when `\rotate` has been used (in the cell where the command `\Block` is used but maybe in `#4`, `\RowStyle`, `code-for-first-row`, etc.).

```

7358         \@@_adjust_hpos_rotate:

```

The boolean `\g_@@_rotate_bool` will be also considered *after the composition of the box* (in order to rotate the box).

Remind that we are in the command of composition of the box of the block. Previously, we have only done some tuning. Now, we will actually compose the content with a `{tabular}`, an `{array}` or a `{minipage}`.

```

7359         \bool_if:NTF \l_@@_tabular_bool
7360         {

```

```

7361     \bool_lazy_all:nTF
7362     {
7363     { \int_compare_p:nNn { #2 } = { \c_one_int } }

```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of -1 cm.

```

7364     {
7365     ! \dim_compare_p:nNn
7366     { \l_@@_col_width_dim } < { \c_zero_dim }
7367     }
7368     { ! \g_@@_rotate_bool }
7369     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7370     {
7371     \use:e
7372     {

```

The `\exp_not:N` is mandatory before `\begin`. It will be possible to delete the `\exp_not:N` in TeXLive 2025 because `\begin` is now protected by `\protected` (and not by `\protect`). There is several other occurrences in that document.

```

7373     \exp_not:N \begin { minipage }
7374     [ \str_lowercase:f \l_@@_vpos_block_str ]
7375     { \l_@@_col_width_dim }
7376     \str_case:on \l_@@_hpos_block_str
7377     { c \centering r \raggedleft l \raggedright }
7378     }
7379     #5
7380     \end { minipage }
7381     }

```

In the other cases, we use a `{tabular}`.

```

7382     {
7383     \bool_if:NT \c_@@_testphase_table_bool
7384     { \tagpdfsetup { table / tagging = presentation } }
7385     \use:e
7386     {
7387     \exp_not:N \begin { tabular }
7388     [ \str_lowercase:f \l_@@_vpos_block_str ]
7389     { @ { } \l_@@_hpos_block_str @ { } }
7390     }
7391     #5
7392     \end { tabular }
7393     }
7394     }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7395     {
7396     \c_math_toggle_token
7397     \use:e
7398     {
7399     \exp_not:N \begin { array }
7400     [ \str_lowercase:f \l_@@_vpos_block_str ]
7401     { @ { } \l_@@_hpos_block_str @ { } }
7402     }
7403     #5
7404     \end { array }
7405     \c_math_toggle_token
7406     }
7407     }

```

The box which will contain the content of the block has now been composed.

If there were `\rotate` (which raises `\g_@@_rotate_bool`) in the content of the `\Block`, we do a rotation of the box (and we also adjust the baseline of the rotated box).

```
7408 \bool_if:NT \g_@@_rotate_bool { \@@_rotate_box_of_block: }
```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```
7409 \int_compare:nNnT { #2 } = { \c_one_int }
7410 {
7411   \dim_gset:Nn \g_@@_blocks_wd_dim
7412   {
7413     \dim_max:nn
7414     { \g_@@_blocks_wd_dim }
7415     {
7416       \box_wd:c
7417       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7418     }
7419   }
7420 }
```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position T or B. Remind that if the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```
7421 \int_compare:nNnT { #1 } = { \c_one_int }
7422 {
7423   \bool_lazy_any:nT
7424   {
7425     { \str_if_empty_p:N \l_@@_vpos_block_str }
7426     { \str_if_eq_p:ee \l_@@_vpos_block_str { t } }
7427     { \str_if_eq_p:ee \l_@@_vpos_block_str { b } }
7428   }
7429   { \@@_adjust_blocks_ht_dp: }
7430 }
7431 \seq_gput_right:Ne \g_@@_blocks_seq
7432 {
7433   \l_tmpa_tl
```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in `\l_@@_hpos_block_str`. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of `\l_@@_hpos_block_str`, which is fixed by the type of current column.

```
7434 {
7435   \exp_not:n { #3 } ,
7436   \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
7437 \bool_if:NT \g_@@_rotate_bool
7438 {
7439   \bool_if:NTF \g_@@_rotate_c_bool
7440   { m }
7441   {
7442     \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
7443     { T }
7444   }
7445 }
7446 {
7447   {
7448     \box_use_drop:c
7449     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7450   }
7451 }
7452 \bool_set_false:N \g_@@_rotate_c_bool
7453 }
7454 \cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
7455 {
7456   \dim_gset:Nn \g_@@_blocks_ht_dim
```



```

7457 {
7458   \dim_max:nn
7459   { \g_@@_blocks_ht_dim }
7460   {
7461     \box_ht:c
7462     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7463   }
7464 }
7465 \dim_gset:Nn \g_@@_blocks_dp_dim
7466 {
7467   \dim_max:nn
7468   { \g_@@_blocks_dp_dim }
7469   {
7470     \box_dp:c
7471     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7472   }
7473 }
7474 }

7475 \cs_new:Npn \@@_adjust_hpos_rotate:
7476 {
7477   \bool_if:NT \g_@@_rotate_bool
7478   {
7479     \str_set:Ne \l_@@_hpos_block_str
7480     {
7481       \bool_if:NTF \g_@@_rotate_c_bool
7482       { c }
7483       {
7484         \str_case:onF \l_@@_vpos_block_str
7485         { b l B l t r T r }
7486         {
7487           \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7488           { r }
7489           { l }
7490         }
7491       }
7492     }
7493   }
7494 }
7495 \cs_generate_variant:Nn \@@_Block_iv:nmnn { e e }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7496 \cs_new_protected:Npn \@@_rotate_box_of_block:
7497 {
7498   \box_grotate:cn
7499   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7500   { 90 }
7501   \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
7502   {
7503     \vbox_gset_top:cn
7504     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7505     {
7506       \skip_vertical:n { 0.8 ex }
7507       \box_use:c
7508       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7509     }
7510   }
7511   \bool_if:NT \g_@@_rotate_c_bool
7512   {
7513     \hbox_gset:cn
7514     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

```

7515     {
7516     \c_math_toggle_token
7517     \vcenter
7518     {
7519     \box_use:c
7520     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7521     }
7522     \c_math_toggle_token
7523     }
7524 }
7525 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7526 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7527 {
7528 \seq_gput_right:Ne \g_@@_blocks_seq
7529 {
7530 \l_tmpa_tl
7531 { \exp_not:n { #3 } }
7532 {
7533 \bool_if:NTF \l_@@_tabular_bool
7534 {
7535 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7536 \@@_reset_arraystretch:
7537 \exp_not:n
7538 {
7539 \dim_zero:N \extrarowheight
7540 #4

```

If the box is rotated (the key `\rotate` may be in the previous #4), the tabular used for the content of the cell will be constructed with a format `c`. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

```

7541 \bool_if:NT \c_@@_testphase_table_bool
7542 { \tag_stop:n { table } }
7543 \use:e
7544 {
7545 \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7546 { @ { } \l_@@_hpos_block_str @ { } }
7547 }
7548 #5
7549 \end { tabular }
7550 }
7551 \group_end:
7552 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7553 {
7554 \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7555 \@@_reset_arraystretch:
7556 \exp_not:n
7557 {
7558 \dim_zero:N \extrarowheight

```

```

7559         #4
7560         \c_math_toggle_token
7561         \use:e
7562         {
7563             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7564             { @ { } \l_@@_hpos_block_str @ { } }
7565         }
7566         #5
7567         \end { array }
7568         \c_math_toggle_token
7569     }
7570     \group_end:
7571 }
7572 }
7573 }
7574 }
7575 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7576 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7577 {
7578     \seq_gput_right:Ne \g_@@_blocks_seq
7579     {
7580         \l_tmpa_tl
7581         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7582         { { \exp_not:n { #4 #5 } } }
7583     }
7584 }
7585 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }

```

The following macro is also for the case of a `\Block` which uses the key `p`.

```

7586 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7587 {
7588     \seq_gput_right:Ne \g_@@_blocks_seq
7589     {
7590         \l_tmpa_tl
7591         { \exp_not:n { #3 } }
7592         { \exp_not:n { #4 #5 } }
7593     }
7594 }
7595 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }

```

We recall that the options of the command `\Block` are analyzed twice: first in the cell of the array and once again when the block will be put in the array *after the construction of the array* (by using PGF).

```

7596 \keys_define:nn { nicematrix / Block / SecondPass }
7597 {
7598     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7599     ampersand-in-blocks .default:n = true ,
7600     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7601     tikz .code:n =
7602         \IfPackageLoadedTF { tikz }
7603             { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7604             { \@@_error:n { tikz~key~without~tikz } } ,
7605     tikz .value_required:n = true ,
7606     fill .code:n =
7607         \tl_set_rescan:Nnn

```

```

7608     \l_@@_fill_tl
7609     { \char_set_catcode_other:N ! }
7610     { #1 } ,
7611 fill .value_required:n = true ,
7612 opacity .tl_set:N = \l_@@_opacity_tl ,
7613 opacity .value_required:n = true ,
7614 draw .code:n =
7615     \tl_set_rescan:Nnn
7616     \l_@@_draw_tl
7617     { \char_set_catcode_other:N ! }
7618     { #1 } ,
7619 draw .default:n = default ,
7620 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7621 rounded-corners .default:n = 4 pt ,
7622 color .code:n =
7623     \@@_color:n { #1 }
7624     \tl_set_rescan:Nnn
7625     \l_@@_draw_tl
7626     { \char_set_catcode_other:N ! }
7627     { #1 } ,
7628 borders .clist_set:N = \l_@@_borders_clist ,
7629 borders .value_required:n = true ,
7630 hvlines .meta:n = { vlines , hlines } ,
7631 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7632 vlines .default:n = true ,
7633 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7634 hlines .default:n = true ,
7635 line-width .dim_set:N = \l_@@_line_width_dim ,
7636 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7637 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7638     \bool_set_true:N \l_@@_p_block_bool ,
7639 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7640 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7641 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7642 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7643     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7644 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7645     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7646 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7647     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7648 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7649 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7650 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7651 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7652 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7653 m .value_forbidden:n = true ,
7654 v-center .meta:n = m ,
7655 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7656 p .value_forbidden:n = true ,
7657 name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
7658 name .value_required:n = true ,
7659 name .initial:n = ,
7660 respect-arraystretch .code:n =
7661     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7662 respect-arraystretch .value_forbidden:n = true ,
7663 transparent .bool_set:N = \l_@@_transparent_bool ,
7664 transparent .default:n = true ,
7665 transparent .initial:n = false ,
7666 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7667 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construc-

tion of the array. We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

7668 \cs_new_protected:Npn \@@_draw_blocks:
7669 {
7670   \bool_if:nTF { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
7671     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7672     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7673   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7674 }
7675 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7676 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7677   \int_zero:N \l_@@_last_row_int
7678   \int_zero:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as follows: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That’s what we detect now (we write 98 for the case the the command `\Block` has been issued in the “first row”).

```

7679   \int_compare:nNnTF { #3 } > { 98 }
7680     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7681     { \int_set:Nn \l_@@_last_row_int { #3 } }
7682   \int_compare:nNnTF { #4 } > { 98 }
7683     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7684     { \int_set:Nn \l_@@_last_col_int { #4 } }
7685   \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
7686     {
7687       \bool_lazy_and:nnTF
7688         { \l_@@_preamble_bool }
7689         {
7690           \int_compare_p:n
7691             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7692         }
7693       {
7694         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7695         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7696         \@@_msg_redirect_name:nn { columns-not-used } { none }
7697       }
7698       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7699     }
7700   {
7701     \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
7702       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7703       {
7704         \@@_Block_v:nneenn
7705           { #1 }
7706           { #2 }
7707           { \int_use:N \l_@@_last_row_int }
7708           { \int_use:N \l_@@_last_col_int }
7709           { #5 }
7710           { #6 }
7711       }
7712     }
7713 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. `#1` is the first row of the block; `#2` is the first column of the block; `#3` is the last row of the block; `#4` is the last column of the block; `#5` is a list of `key=value` options; `#6` is the label

```

7714 \cs_new_protected:Npn \l_@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7715 {

```

The group is for the keys.

```

7716 \group_begin:
7717 \int_compare:nNnT { #1 } = { #3 }
7718 { \str_set:Nn \l_@@_vpos_block_str { t } }
7719 \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains &, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that \tl_if_in:nnT is faster than \str_if_in:nnT.

```

7720 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7721 \bool_lazy_and:nnT
7722 { \l_@@_vlines_block_bool }
7723 { ! \l_@@_ampersand_bool }
7724 {
7725 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7726 {
7727 \l_@@_vlines_block:nnn
7728 { \exp_not:n { #5 } }
7729 { #1 - #2 }
7730 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7731 }
7732 }
7733 \bool_if:NT \l_@@_hlines_block_bool
7734 {
7735 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7736 {
7737 \l_@@_hlines_block:nnn
7738 { \exp_not:n { #5 } }
7739 { #1 - #2 }
7740 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7741 }
7742 }
7743 \bool_if:NF \l_@@_transparent_bool
7744 {
7745 \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
7746 {

```

The sequence of the positions of the blocks (excepted the blocks with the key hvlines) will be used when drawing the rules (in fact, there is also the \multicolumn and the \diagbox in that sequence).

```

7747 \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7748 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7749 }
7750 }

```

```

7751 \tl_if_empty:NF \l_@@_draw_tl
7752 {
7753 \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7754 { \@@_error:n { hlines~with~color } }
7755 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7756 {
7757 \l_@@_stroke_block:nnn

```

#5 are the options

```

7758 { \exp_not:n { #5 } }
7759 { #1 - #2 }
7760 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7761 }
7762 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7763 { { #1 } { #2 } { #3 } { #4 } }
7764 }

```

```

7765 \clist_if_empty:NF \l_@@_borders_clist
7766 {
7767   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7768   {
7769     \@@_stroke_borders_block:nnn
7770     { \exp_not:n { #5 } }
7771     { #1 - #2 }
7772     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7773   }
7774 }
7775 \tl_if_empty:NF \l_@@_fill_tl
7776 {
7777   \@@_add_opacity_to_fill:
7778   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7779   {
7780     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7781     { #1 - #2 }
7782     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7783     { \dim_use:N \l_@@_rounded_corners_dim }
7784   }
7785 }
7786 \seq_if_empty:NF \l_@@_tikz_seq
7787 {
7788   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7789   {
7790     \@@_block_tikz:nnnnn
7791     { \seq_use:Nn \l_@@_tikz_seq { , } }
7792     { #1 }
7793     { #2 }
7794     { \int_use:N \l_@@_last_row_int }
7795     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7796   }
7797 }
7798 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7799 {
7800   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7801   {
7802     \@@_actually_diagbox:nnnnnn
7803     { #1 }
7804     { #2 }
7805     { \int_use:N \l_@@_last_row_int }
7806     { \int_use:N \l_@@_last_col_int }
7807     { \exp_not:n { ##1 } }
7808     { \exp_not:n { ##2 } }
7809   }
7810 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & & four & five & \\\
six & & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node 1-1-block

our block		one
three	four	two
six	seven	five
		eight

We highlight the node 1-1-block-short

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7811 \pgfpicture
7812 \pgfrememberpicturepositiononpagetrue
7813 \pgf@relevantforpicturesizefalse
7814 \@@_qpoint:n { row - #1 }
7815 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7816 \@@_qpoint:n { col - #2 }
7817 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7818 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7819 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7820 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7821 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

We construct the node for the block with the name (#1-#2-block).

The function `\@@_pgf_rect_node:nnnnn` takes in as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```

7822 \@@_pgf_rect_node:nnnnn
7823 { \@@_env: - #1 - #2 - block }
7824 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7825 \str_if_empty:NF \l_@@_block_name_str
7826 {
7827 \pgfnodealias
7828 { \@@_env: - \l_@@_block_name_str }
7829 { \@@_env: - #1 - #2 - block }
7830 \str_if_empty:NF \l_@@_name_str
7831 {
7832 \pgfnodealias
7833 { \l_@@_name_str - \l_@@_block_name_str }
7834 { \@@_env: - #1 - #2 - block }
7835 }
7836 }

```

Now, we create the “short node” which, in general, will be used to put the label (that is to say the content of the node). However, if one the keys L, C or R is used (that information is provided by the boolean `\l_@@_hpos_of_block_cap_bool`), we don’t need to create that node since the normal node is used to put the label.

```

7837 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7838 {
7839 \dim_set_eq:NN \l_tmpb_dim \c_max_dim

```

The short node is constructed by taking into account the *contents* of the columns involved in at least one cell of the block. That’s why we have to do a loop over the rows of the array.

```

7840 \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7841 {

```

We recall that, when a cell is empty, no (normal) node is created in that cell. That’s why we test the existence of the node before using it.

```

7842 \cs_if_exist:cT
7843 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7844 {
7845 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7846 {
7847 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7848 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7849 }
7850 }
7851 }

```


If all the cells of the column were empty, `\l_tmpb_dim` has still the same value `\c_max_dim`. In that case, you use for `\l_tmpb_dim` the value of the position of the vertical rule.

```

7852     \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
7853     {
7854         \@@_qpoint:n { col - #2 }
7855         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7856     }
7857     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7858     \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
7859     {
7860         \cs_if_exist:cT
7861         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7862         {
7863             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7864             {
7865                 \pgfpointanchor
7866                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7867                 { east }
7868                 \dim_set:Nn \l_@@_tmpd_dim
7869                 { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
7870             }
7871         }
7872     }
7873     \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
7874     {
7875         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7876         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7877     }
7878     \@@_pgf_rect_node:nnnn
7879     { \@@_env: - #1 - #2 - block - short }
7880     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7881 }

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

7882     \bool_if:NT \l_@@_medium_nodes_bool
7883     {
7884         \@@_pgf_rect_node:nnn
7885         { \@@_env: - #1 - #2 - block - medium }
7886         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7887         {
7888             \pgfpointanchor
7889             { \@@_env:
7890               - \int_use:N \l_@@_last_row_int
7891               - \int_use:N \l_@@_last_col_int - medium
7892             }
7893             { south-east }
7894         }
7895     }
7896     \endpgfpicture
7897
7898     \bool_if:NTF \l_@@_ampersand_bool
7899     {
7900         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7901         \int_zero_new:N \l_@@_split_int
7902         \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7903         \pgfpicture
7904         \pgfrememberpicturepositiononpagetrue
7905         \pgf@relevantforpicturesizefalse
7906
7907         \@@_qpoint:n { row - #1 }

```

```

7908 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7909 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7910 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7911 \@@_qpoint:n { col - #2 }
7912 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7913 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7914 \dim_set:Nn \l_tmpb_dim
7915 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7916 \bool_lazy_or:nnT
7917 { \l_@@_vlines_block_bool }
7918 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7919 {
7920 \int_step_inline:nn { \l_@@_split_int - 1 }
7921 {
7922 \pgfpathmoveto
7923 {
7924 \pgfpoint
7925 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7926 \l_@@_tmpc_dim
7927 }
7928 \pgfpathlineto
7929 {
7930 \pgfpoint
7931 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7932 \l_@@_tmpd_dim
7933 }
7934 \CT@arc@
7935 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7936 \pgfsetrectcap
7937 \pgfusepathqstroke
7938 }
7939 }
7940 \@@_qpoint:n { row - #1 - base }
7941 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7942 \int_step_inline:nn { \l_@@_split_int }
7943 {
7944 \group_begin:
7945 \dim_set:Nn \col@sep
7946 { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
7947 \pgftransformshift
7948 {
7949 \pgfpoint
7950 {
7951 \l_tmpa_dim + ##1 \l_tmpb_dim -
7952 \str_case:on \l_@@_hpos_block_str
7953 {
7954 l { \l_tmpb_dim + \col@sep }
7955 c { 0.5 \l_tmpb_dim }
7956 r { \col@sep }
7957 }
7958 }
7959 { \l_@@_tmpc_dim }
7960 }
7961 \pgfset { inner~sep = \c_zero_dim }
7962 \pgfnode
7963 { rectangle }
7964 {
7965 \str_case:on \l_@@_hpos_block_str
7966 {
7967 c { base }
7968 l { base-west }
7969 r { base-east }
7970 }

```

```

7971     }
7972     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7973     \group_end:
7974   }
7975   \endpgfpicture
7976 }

```

Now the case where there is no ampersand & in the content of the block.

```

7977 {
7978   \bool_if:NTF \l_@@_p_block_bool
7979   {

```

When the final user has used the key p, we have to compute the width.

```

7980     \pgfpicture
7981     \pgfrememberpicturepositiononpagetrue
7982     \pgf@relevantforpicturesizefalse
7983     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7984     {
7985       \@@_qpoint:n { col - #2 }
7986       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7987       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7988     }
7989     {
7990       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7991       \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7992       \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7993     }
7994     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7995   \endpgfpicture
7996   \hbox_set:Nn \l_@@_cell_box
7997   {
7998     \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
7999     { \g_tmpb_dim }
8000     \str_case:on \l_@@_hpos_block_str
8001     { c \centering r \raggedleft l \raggedright j { } }
8002     #6
8003     \end { minipage }
8004   }
8005 }
8006 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
8007 \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }

```

Now, we will put the label of the block. We recall that \l_@@_vpos_block_str is empty when the user has not used a key for the vertical position of the block.

```

8008   \pgfpicture
8009   \pgfrememberpicturepositiononpagetrue
8010   \pgf@relevantforpicturesizefalse
8011   \bool_lazy_any:nTF
8012   {
8013     { \str_if_empty_p:N \l_@@_vpos_block_str }
8014     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
8015     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
8016     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
8017   }
8018   {

```

If we are in the first column, we must put the block as if it was with the key r.

```

8019     \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the last column, we must put the block as if it was with the key l.

```

8020     \bool_if:nT \g_@@_last_col_found_bool
8021     {
8022       \int_compare:nNnT { #2 } = { \g_@@_col_total_int }

```

```

8023         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
8024     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

8025     \tl_set:Ne \l_tmpa_tl
8026     {
8027         \str_case:on \l_@@_vpos_block_str
8028         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

8029         { } {
8030             \str_case:on \l_@@_hpos_block_str
8031             {
8032                 c { center }
8033                 l { west }
8034                 r { east }
8035                 j { center }
8036             }
8037         }
8038     c {
8039         \str_case:on \l_@@_hpos_block_str
8040         {
8041             c { center }
8042             l { west }
8043             r { east }
8044             j { center }
8045         }
8046     }
8047     T {
8048         \str_case:on \l_@@_hpos_block_str
8049         {
8050             c { north }
8051             l { north~west }
8052             r { north~east }
8053             j { north }
8054         }
8055     }
8056     B {
8057         \str_case:on \l_@@_hpos_block_str
8058         {
8059             c { south }
8060             l { south~west }
8061             r { south~east }
8062             j { south }
8063         }
8064     }
8065     }
8066     }
8067     }
8068     }
8069     }
8070 \pgftransformshift
8071 {
8072     \pgfpointanchor
8073     {
8074         \@@_env: - #1 - #2 - block
8075         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8076     }
8077     { \l_tmpa_tl }
8078 }
8079 \pgfset { inner~sep = \c_zero_dim }
8080 \pgfnode
8081 { rectangle }

```

```

8082     { \l_tmpa_tl }
8083     { \box_use_drop:N \l_@@_cell_box } { } { }
8084 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

8085 {
8086   \pgfextracty \l_tmpa_dim
8087   {
8088     \@@_qpoint:n
8089     {
8090       row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8091       - base
8092     }
8093   }
8094   \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8095   \pgfpointanchor
8096   {
8097     \@@_env: - #1 - #2 - block
8098     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8099   }
8100   {
8101     \str_case:on \l_@@_hpos_block_str
8102     {
8103       c { center }
8104       l { west }
8105       r { east }
8106       j { center }
8107     }
8108   }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8109   \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8110   \pgfset { inner~sep = \c_zero_dim }
8111   \pgfnode
8112   { rectangle }
8113   {
8114     \str_case:on \l_@@_hpos_block_str
8115     {
8116       c { base }
8117       l { base-west }
8118       r { base-east }
8119       j { base }
8120     }
8121   }
8122   { \box_use_drop:N \l_@@_cell_box } { } { }
8123 }
8124 \endpgfpicture
8125 }
8126 \group_end:
8127 }
8128 \cs_generate_variant:Nn \@@_Block_v:nnnnn { n n e e }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8129 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8130 {
8131   \pgfpicture
8132   \pgfrememberpicturepositiononpagetrue
8133   \pgf@relevantforpicturesizefalse
8134   \pgfpathrectanglecorners
8135   { \pgfpoint { #2 } { #3 } }

```

```

8136     { \pgfpoint { #4 } { #5 } }
8137   \pgfsetfillcolor { #1 }
8138   \pgfusepath { fill }
8139   \endpgfpicture
8140 }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

8141 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8142 {
8143   \tl_if_empty:NF \l_@@_opacity_tl
8144   {
8145     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8146       {
8147         \tl_set:Ne \l_@@_fill_tl
8148         {
8149           [ opacity = \l_@@_opacity_tl ,
8150           \tl_tail:o \l_@@_fill_tl
8151         }
8152       }
8153       {
8154         \tl_set:Ne \l_@@_fill_tl
8155         { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8156       }
8157     }
8158 }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax *i-j*) and the third is the last cell of the block (with the same syntax).

```

8159 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8160 {
8161   \group_begin:
8162   \tl_clear:N \l_@@_draw_tl
8163   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8164   \keys_set_known:nm { nicematrix / BlockStroke } { #1 }
8165   \pgfpicture
8166   \pgfrememberpicturepositiononpagetrue
8167   \pgf@relevantforpicturesizefalse
8168   \tl_if_empty:NF \l_@@_draw_tl
8169   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8170     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8171     { \CT@arc@ }
8172     { \@@_color:o \l_@@_draw_tl }
8173   }
8174   \pgfsetcornersarced
8175   {
8176     \pgfpoint
8177     { \l_@@_rounded_corners_dim }
8178     { \l_@@_rounded_corners_dim }
8179   }
8180   \@@_cut_on_hyphen:w #2 \q_stop
8181   \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
8182   {
8183     \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
8184     {
8185       \@@_qpoint:n { row - \l_tmpa_tl }
8186       \dim_set_eq:NN \l_tmpb_dim \pgf@y

```

```

8187     \@@_qpoint:n { col - \l_tmpb_tl }
8188     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8189     \@@_cut_on_hyphen:w #3 \q_stop
8190     \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
8191       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8192     \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
8193       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8194     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8195     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8196     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8197     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8198     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8199     \pgfpathrectanglecorners
8200       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8201       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8202     \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
8203       { \pgfusepathqstroke }
8204       { \pgfusepath { stroke } }
8205   }
8206 }
8207 \endpgfpicture
8208 \group_end:
8209 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8210 \keys_define:nn { nicematrix / BlockStroke }
8211 {
8212   color .tl_set:N = \l_@@_draw_tl ,
8213   draw .code:n =
8214     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8215   draw .default:n = default ,
8216   line-width .dim_set:N = \l_@@_line_width_dim ,
8217   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8218   rounded-corners .default:n = 4 pt
8219 }

```

The first argument of `\@@_vlines_block:nnn` is a list of options for the rules that we will draw. The second argument is the upper-left cell of the block (with, as usual, the syntax i - j) and the third is the last cell of the block (with the same syntax).

```

8220 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8221 {
8222   \group_begin:
8223   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8224   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8225   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8226   \@@_cut_on_hyphen:w #2 \q_stop
8227   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8228   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8229   \@@_cut_on_hyphen:w #3 \q_stop
8230   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8231   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8232   \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
8233   {
8234     \use:e
8235     {
8236       \@@_vline:n
8237       {
8238         position = ##1 ,
8239         start = \l_@@_tmpc_tl ,
8240         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8241         total-width = \dim_use:N \l_@@_line_width_dim
8242       }
8243     }

```

```

8244     }
8245   \group_end:
8246 }

8247 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8248 {
8249   \group_begin:
8250   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8251   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8252   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8253   \@@_cut_on_hyphen:w #2 \q_stop
8254   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8255   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8256   \@@_cut_on_hyphen:w #3 \q_stop
8257   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8258   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8259   \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
8260   {
8261     \use:e
8262     {
8263       \@@_hline:n
8264       {
8265         position = ##1 ,
8266         start = \l_@@_tmpd_tl ,
8267         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8268         total-width = \dim_use:N \l_@@_line_width_dim
8269       }
8270     }
8271   }
8272   \group_end:
8273 }

```

The first argument of `\@@_stroke_borders_block:nnn` is a list of options for the borders that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

```

8274 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8275 {
8276   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8277   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8278   \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
8279   { \@@_error:n { borders~forbidden } }
8280   {
8281     \tl_clear_new:N \l_@@_borders_tikz_tl
8282     \keys_set:no
8283     { nicematrix / OnlyForTikzInBorders }
8284     \l_@@_borders_clist
8285     \@@_cut_on_hyphen:w #2 \q_stop
8286     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8287     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8288     \@@_cut_on_hyphen:w #3 \q_stop
8289     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8290     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8291     \@@_stroke_borders_block_i:
8292   }
8293 }

8294 \hook_gput_code:nnn { begindocument } { . }
8295 {
8296   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8297   {
8298     \c_@@_pgfortikzpicture_tl
8299     \@@_stroke_borders_block_ii:
8300     \c_@@_endpgfortikzpicture_tl
8301   }

```



```

8302 }
8303 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8304 {
8305   \pgfrememberpicturepositiononpagetrue
8306   \pgf@relevantforpicturesizefalse
8307   \CT@arc@
8308   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8309   \clist_if_in:NnT \l_@@_borders_clist { right }
8310     { \@@_stroke_vertical:n \l_tmpb_tl }
8311   \clist_if_in:NnT \l_@@_borders_clist { left }
8312     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8313   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8314     { \@@_stroke_horizontal:n \l_tmpa_tl }
8315   \clist_if_in:NnT \l_@@_borders_clist { top }
8316     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8317 }
8318 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8319 {
8320   tikz .code:n =
8321     \cs_if_exist:NTF \tikzpicture
8322       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8323       { \@@_error:n { tikz-in-borders-without-tikz } } ,
8324   tikz .value_required:n = true ,
8325   top .code:n = ,
8326   bottom .code:n = ,
8327   left .code:n = ,
8328   right .code:n = ,
8329   unknown .code:n = \@@_error:n { bad-border }
8330 }

```

The following command is used to stroke the left border and the right border. The argument #1 is the number of column (in the sense of the col node).

```

8331 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8332 {
8333   \@@_qpoint:n \l_@@_tmpc_tl
8334   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8335   \@@_qpoint:n \l_tmpa_tl
8336   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8337   \@@_qpoint:n { #1 }
8338   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8339     {
8340       \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8341       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8342       \pgfusepathqstroke
8343     }
8344     {
8345       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8346         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8347     }
8348 }

```

The following command is used to stroke the top border and the bottom border. The argument #1 is the number of row (in the sense of the row node).

```

8349 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8350 {
8351   \@@_qpoint:n \l_@@_tmpd_tl
8352   \clist_if_in:NnTF \l_@@_borders_clist { left }
8353     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8354     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8355   \@@_qpoint:n \l_tmpb_tl
8356   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8357   \@@_qpoint:n { #1 }

```

```

8358 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8359 {
8360   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8361   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8362   \pgfusepathqstroke
8363 }
8364 {
8365   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8366   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8367 }
8368 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8369 \keys_define:nn { nicematrix / BlockBorders }
8370 {
8371   borders .clist_set:N = \l_@@_borders_clist ,
8372   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8373   rounded-corners .default:n = 4 pt ,
8374   line-width .dim_set:N = \l_@@_line_width_dim
8375 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`. `#1` is a *list of lists* of Tikz keys used with the path.

Example: `{\offset=1pt,draw,red},{\offset=2pt,draw,blue}`

which arises from a command such as :

`\Block[tikz={\offset=1pt,draw,red},tikz={\offset=2pt,draw,blue}]{2-2}{}`

The arguments `#2` and `#3` are the coordinates of the first cell and `#4` and `#5` the coordinates of the last cell of the block.

```

8376 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8377 {
8378   \begin { tikzpicture }
8379   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8380   \clist_map_inline:nn { #1 }
8381   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8382     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8383     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8384     (
8385     [
8386       xshift = \dim_use:N \l_@@_offset_dim ,
8387       yshift = - \dim_use:N \l_@@_offset_dim
8388     ]
8389     #2 -| #3
8390     )
8391     rectangle
8392     (
8393     [
8394       xshift = - \dim_use:N \l_@@_offset_dim ,
8395       yshift = \dim_use:N \l_@@_offset_dim
8396     ]
8397     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8398     ) ;
8399   }
8400   \end { tikzpicture }
8401 }
8402 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

```

```

8403 \keys_define:nn { nicematrix / SpecialOffset }
8404 { offset .dim_set:N = \l_@@_offset_dim }

```

In some circumstances, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock`: which has the same syntax as the standard command `\Block` but which is no-op.

```

8405 \cs_new_protected:Npn \@@_NullBlock:
8406   { \@@_collect_options:n { \@@_NullBlock_i: } }
8407 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8408   { }

```

27 How to draw the dotted lines transparently

```

8409 \cs_set_protected:Npn \@@_renew_matrix:
8410   {
8411     \RenewDocumentEnvironment { pmatrix } { }
8412     { \pNiceMatrix }
8413     { \endpNiceMatrix }
8414     \RenewDocumentEnvironment { vmatrix } { }
8415     { \vNiceMatrix }
8416     { \endvNiceMatrix }
8417     \RenewDocumentEnvironment { Vmatrix } { }
8418     { \VNiceMatrix }
8419     { \endVNiceMatrix }
8420     \RenewDocumentEnvironment { bmatrix } { }
8421     { \bNiceMatrix }
8422     { \endbNiceMatrix }
8423     \RenewDocumentEnvironment { Bmatrix } { }
8424     { \BNiceMatrix }
8425     { \endBNiceMatrix }
8426   }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8427 \keys_define:nn { nicematrix / Auto }
8428   {
8429     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8430     columns-type .value_required:n = true ,
8431     l .meta:n = { columns-type = l } ,
8432     r .meta:n = { columns-type = r } ,
8433     c .meta:n = { columns-type = c } ,
8434     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8435     delimiters / color .value_required:n = true ,
8436     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8437     delimiters / max-width .default:n = true ,
8438     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8439     delimiters .value_required:n = true ,
8440     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8441     rounded-corners .default:n = 4 pt
8442   }
8443 \NewDocumentCommand \AutoNiceMatrixWithDelims
8444   { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8445   { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8446 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8447   {

```

The group is for the protection of the keys.

```

8448   \group_begin:
8449   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl

```

```

8450 \use:e
8451 {
8452   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8453   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8454   [ \exp_not:o \l_tmpa_tl ]
8455 }
8456 \int_if_zero:nT { \l_@@_first_row_int }
8457 {
8458   \int_if_zero:nT { \l_@@_first_col_int } { & }
8459   \prg_replicate:nn { #4 - 1 } { & }
8460   \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8461 }
8462 \prg_replicate:nn { #3 }
8463 {
8464   \int_if_zero:nT { \l_@@_first_col_int } { & }

```

We put { } before #6 to avoid a hasty expansion of a potential \arabic{iRow} at the beginning of the row which would result in an incorrect value of that iRow (since iRow is incremented in the first cell of the row of the \halign).

```

8465   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8466   \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8467 }
8468 \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
8469 {
8470   \int_if_zero:nT { \l_@@_first_col_int } { & }
8471   \prg_replicate:nn { #4 - 1 } { & }
8472   \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
8473 }
8474 \end { NiceArrayWithDelims }
8475 \group_end:
8476 }
8477 \cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
8478 {
8479   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8480   {
8481     \bool_gset_true:N \g_@@_delims_bool
8482     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8483     \AutoNiceMatrixWithDelims { #2 } { #3 }
8484   }
8485 }

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8486 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8487 {
8488   \group_begin:
8489   \bool_gset_false:N \g_@@_delims_bool
8490   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8491   \group_end:
8492 }

```

29 The redefinition of the command \dotfill

```

8493 \cs_set_eq:NN \@@_old_dotfill: \dotfill
8494 \cs_new_protected:Npn \@@_dotfill:
8495 {

```

First, we insert \@@_dotfill (which is the saved version of \dotfill) in case of use of \dotfill “internally” in the cell (e.g. \hbox to 1cm {\dotfill}).

```

8496   \@@_old_dotfill:

```

```

8497     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8498   }

```

Now, if the box is not empty (unfortunately, we can't actually test whether the box is empty and that's why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8499 \cs_new_protected:Npn \@@_dotfill_i:
8500 {
8501   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
8502     { \@@_old_dotfill: }
8503 }

```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8504 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8505 {
8506   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8507     {
8508     \@@_actually_diagbox:nnnnnn
8509     { \int_use:N \c@iRow }
8510     { \int_use:N \c@jCol }
8511     { \int_use:N \c@iRow }
8512     { \int_use:N \c@jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```

8513     { \g_@@_row_style_tl \exp_not:n { #1 } }
8514     { \g_@@_row_style_tl \exp_not:n { #2 } }
8515   }

```

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key corners.

```

8516   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8517     {
8518     { \int_use:N \c@iRow }
8519     { \int_use:N \c@jCol }
8520     { \int_use:N \c@iRow }
8521     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8522     { }
8523   }
8524 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

8525 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8526 {
8527   \pgfpicture
8528   \pgf@relevantforpicturesizefalse
8529   \pgfrememberpicturepositiononpagetrue
8530   \@@_qpoint:n { row - #1 }
8531   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8532   \@@_qpoint:n { col - #2 }

```

```

8533 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8534 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8535 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8536 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8537 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8538 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8539 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8540 {

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. The package `nicematrix` uses it even if `colortbl` is not loaded.

```

8541 \CT@arc@
8542 \pgfsetroundcap
8543 \pgfusepathqstroke
8544 }
8545 \pgfset { inner~sep = 1 pt }
8546 \pgfscope
8547 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8548 \pgfnode { rectangle } { south~west }
8549 {
8550 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument `#5` is empty.

```

8551 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8552 \end { minipage }
8553 }
8554 { }
8555 { }
8556 \endpgfscope
8557 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8558 \pgfnode { rectangle } { north~east }
8559 {
8560 \begin { minipage } { 20 cm }
8561 \raggedleft
8562 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8563 \end { minipage }
8564 }
8565 { }
8566 { }
8567 \endpgfpicture
8568 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 86.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8569 \cs_new:Npn \@@_CodeAfter: { \omit \@@_CodeAfter_ii:n }

```

However, in each cell of the environment, the command `\CodeAfter` will be linked to the following command `\@@_CodeAfter_ii:n` which begins with `\`.

```

8570 \cs_new_protected:Npn \@@_CodeAfter_i: { \ \omit \@@_CodeAfter_ii:n }

```

We have to catch everything until the end of the current environment (of `nicematrix`). First, we go until the next command `\end`.

```
8571 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8572 {
8573   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8574   \@@_CodeAfter_iv:n
8575 }
```

We catch the argument of the command `\end` (in `#1`).

```
8576 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8577 {
```

If this is really the end of the current environment (of `nicematrix`), we put back the command `\end` and its argument in the TeX flow.

```
8578   \str_if_eq:eeTF \@currenvir { #1 }
8579   { \end { #1 } }
```

If this is not the `\end` we are looking for, we put those tokens in `\g_nicematrix_code_after_tl` and we go on searching for the next command `\end` with a recursive call to the command `\@@_CodeAfter:n`.

```
8580   {
8581     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8582     \@@_CodeAfter_ii:n
8583   }
8584 }
```

32 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
8585 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8586 {
8587   \pgfpicture
8588   \pgfrememberpicturepositiononpagetrue
8589   \pgf@relevantforpicturesizefalse
```

`\l_@@_y_initial_dim` and `\l_@@_y_final_dim` will be the y -values of the extremities of the delimiter we will have to construct.

```
8590   \@@_qpoint:n { row - 1 }
8591   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8592   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8593   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
```

We will compute in `\l_tmpa_dim` the x -value where we will have to put our delimiter (on the left side or on the right side).

```
8594   \bool_if:nTF { #3 }
8595     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8596     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8597   \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
8598     {
8599       \cs_if_exist:cT
8600       { pgf @ sh @ ns @ \@@_env: - #1 - #2 }
```

```

8601     {
8602         \pgfpointanchor
8603         { \@@_env: - #1 - #2 }
8604         { \bool_if:nTF { #3 } { west } { east } }
8605         \dim_set:Nn \l_tmpa_dim
8606         {
8607             \bool_if:nTF { #3 }
8608             { \dim_min:nn }
8609             { \dim_max:nn }
8610             \l_tmpa_dim
8611             { \pgf@x }
8612         }
8613     }
8614 }

```

Now we can put the delimiter with a node of PGF.

```

8615     \pgfset { inner~sep = \c_zero_dim }
8616     \dim_zero:N \nulldelimiterspace
8617     \pgftransformshift
8618     {
8619         \pgfpoint
8620         { \l_tmpa_dim }
8621         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8622     }
8623     \pgfnode
8624     { rectangle }
8625     { \bool_if:nTF { #3 } { east } { west } }
8626     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8627         \nullfont
8628         \c_math_toggle_token
8629         \@@_color:o \l_@@_delimiters_color_tl
8630         \bool_if:nTF { #3 } { \left #1 } { \left . }
8631         \vcenter
8632         {
8633             \nullfont
8634             \hrule \@height
8635                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8636                 \@depth \c_zero_dim
8637                 \@width \c_zero_dim
8638         }
8639         \bool_if:nTF { #3 } { \right . } { \right #1 }
8640         \c_math_toggle_token
8641     }
8642     { }
8643     { }
8644 \endpgfpicture
8645 }

```

33 The command `\SubMatrix`

```

8646 \keys_define:nn { nicematrix / sub-matrix }
8647 {
8648     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8649     extra-height .value_required:n = true ,
8650     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8651     left-xshift .value_required:n = true ,
8652     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8653     right-xshift .value_required:n = true ,
8654     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,

```



```

8655     xshift .value_required:n = true ,
8656     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8657     delimiters / color .value_required:n = true ,
8658     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8659     slim .default:n = true ,
8660     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8661     hlines .default:n = all ,
8662     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8663     vlines .default:n = all ,
8664     hvlines .meta:n = { hlines, vlines } ,
8665     hvlines .value_forbidden:n = true
8666   }
8667 \keys_define:nn { nicematrix }
8668 {
8669   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8670   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8671   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8672   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8673 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8674 \keys_define:nn { nicematrix / SubMatrix }
8675 {
8676   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8677   delimiters / color .value_required:n = true ,
8678   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8679   hlines .default:n = all ,
8680   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8681   vlines .default:n = all ,
8682   hvlines .meta:n = { hlines, vlines } ,
8683   hvlines .value_forbidden:n = true ,
8684   name .code:n =
8685     \tl_if_empty:nTF { #1 }
8686     { \@@_error:n { Invalid-name } }
8687     {
8688       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8689       {
8690         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8691         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8692         {
8693           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8694           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8695         }
8696       }
8697       { \@@_error:n { Invalid-name } }
8698     } ,
8699   name .value_required:n = true ,
8700   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8701   rules .value_required:n = true ,
8702   code .tl_set:N = \l_@@_code_tl ,
8703   code .value_required:n = true ,
8704   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8705 }

8706 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8707 {
8708   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8709   {
8710     \SubMatrix { #1 } { #2 } { #3 } { #4 }
8711     [
8712       delimiters / color = \l_@@_delimiters_color_tl ,
8713       hlines = \l_@@_submatrix_hlines_clist ,

```

```

8714         vlines = \l_@@_submatrix_vlines_clist ,
8715         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8716         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8717         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8718         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8719         #5
8720     ]
8721 }
8722 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8723 \ignorespaces
8724 }
8725 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8726 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8727 { \@@_SubMatrix_in_code_before_i:nmmm #1 #2 }
8728 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nmmm #1 #2 #3 #4
8729 {
8730     \seq_gput_right:Ne \g_@@_submatrix_seq
8731     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8732     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8733     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8734     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8735     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8736 }
8737 }

```

The following macro will compute `\l_@@_first_i_tl`, `\l_@@_first_j_tl`, `\l_@@_last_i_tl` and `\l_@@_last_j_tl` from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

8738 \NewDocumentCommand \@@_compute_i_j:nn
8739 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8740 { \@@_compute_i_j:nmmm #1 #2 }
8741 \cs_new_protected:Npn \@@_compute_i_j:nmmm #1 #2 #3 #4
8742 {
8743     \def \l_@@_first_i_tl { #1 }
8744     \def \l_@@_first_j_tl { #2 }
8745     \def \l_@@_last_i_tl { #3 }
8746     \def \l_@@_last_j_tl { #4 }
8747     \tl_if_eq:NnT \l_@@_first_i_tl { last }
8748     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8749     \tl_if_eq:NnT \l_@@_first_j_tl { last }
8750     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8751     \tl_if_eq:NnT \l_@@_last_i_tl { last }
8752     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8753     \tl_if_eq:NnT \l_@@_last_j_tl { last }
8754     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8755 }

```

In the pre-code-after and in the `\CodeAfter` the following command `\@@_SubMatrix` will be linked to `\SubMatrix`.

- #1 is the left delimiter;
- #2 is the upper-left cell of the matrix with the format $i-j$;
- #3 is the lower-right cell of the matrix with the format $i-j$;
- #4 is the right delimiter;
- #5 is the list of options of the command;
- #6 is the potential subscript;

- #7 is the potential superscript.

For explanations about the construction with rescanning of the preamble, see the documentation for the user command `\Cdots`.

```

8756 \hook_gput_code:nnn { begindocument } { . }
8757 {
8758   \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m 0 { } E { _ ^ } { { } { } } }
8759   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
8760     { \@@_sub_matrix:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
8761 }

8762 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8763 {
8764   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8765   \@@_compute_i_j:nn { #2 } { #3 }
8766   \int_compare:nNt { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
8767     { \def \arraystretch { 1 } }
8768   \bool_lazy_or:nnTF
8769     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
8770     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
8771     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8772     {
8773       \str_clear_new:N \l_@@_submatrix_name_str
8774       \keys_set:nn { nicematrix / SubMatrix } { #5 }
8775       \pgfpicture
8776       \pgfrememberpicturepositiononpagetrue
8777       \pgf@relevantforpicturesizefalse
8778       \pgfset { inner~sep = \c_zero_dim }
8779       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8780       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currfication.

```

8781   \bool_if:NTF \l_@@_submatrix_slim_bool
8782     { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
8783     { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
8784     {
8785       \cs_if_exist:cT
8786         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8787         {
8788           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8789           \dim_compare:nNt { \pgf@x } < { \l_@@_x_initial_dim }
8790             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8791         }
8792       \cs_if_exist:cT
8793         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8794         {
8795           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8796           \dim_compare:nNt { \pgf@x } > { \l_@@_x_final_dim }
8797             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8798         }
8799     }
8800   \dim_compare:nNtTF { \l_@@_x_initial_dim } = { \c_max_dim }
8801     { \@@_error:nn { Impossible-delimiter } { left } }
8802     {
8803       \dim_compare:nNtTF { \l_@@_x_final_dim } = { - \c_max_dim }
8804         { \@@_error:nn { Impossible-delimiter } { right } }
8805         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8806     }
8807   \endpgfpicture
8808 }
8809 \group_end:
8810 \ignorespaces
8811 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8812 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8813 {
8814   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8815   \dim_set:Nn \l_@@_y_initial_dim
8816   {
8817     \fp_to_dim:n
8818     {
8819       \pgf@y
8820       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8821     }
8822   }
8823   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8824   \dim_set:Nn \l_@@_y_final_dim
8825   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8826   \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
8827   {
8828     \cs_if_exist:cT
8829     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8830     {
8831       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8832       \dim_set:Nn \l_@@_y_initial_dim
8833       { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
8834     }
8835     \cs_if_exist:cT
8836     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8837     {
8838       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8839       \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
8840       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8841     }
8842   }
8843   \dim_set:Nn \l_tmpa_dim
8844   {
8845     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8846     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8847   }
8848   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8849   \group_begin:
8850   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8851   \@@_set_CTarc:o \l_@@_rules_color_tl
8852   \CT@arc@

```

Now, we draw the potential vertical rules specified in the preamble of the environments with the letter fixed with the key `vlines-in-sub-matrix`. The list of the columns where there is such rule to draw is in `\g_@@_cols_vlism_seq`.

```

8853   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8854   {
8855     \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
8856     {
8857       \int_compare:nNnT
8858       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8859       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8860         \@@_qpoint:n { col - ##1 }
8861         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8862         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8863         \pgfusepathqstroke
8864       }
8865     }
8866   }

```

Now, we draw the vertical rules specified in the key `vlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

8867   \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8868   { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8869   { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8870   {
8871     \bool_lazy_and:nnTF
8872     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8873     {
8874       \int_compare_p:nNn
8875       { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8876     {
8877       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8878       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8879       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8880       \pgfusepathqstroke
8881     }
8882     { \@@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8883   }

```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

8884   \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8885   { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8886   { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8887   {
8888     \bool_lazy_and:nnTF
8889     { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
8890     {
8891       \int_compare_p:nNn
8892       { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8893     {
8894       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8895   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8896   \dim_set:Nn \l_tmpa_dim
8897   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8898   \str_case:nn { #1 }
8899   {
8900     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8901     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8902     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8903     }
8904     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8905   \dim_set:Nn \l_tmpb_dim
8906   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8907   \str_case:nn { #2 }
8908   {
8909     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8910     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8911     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8912   }
8913   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8914   \pgfusepathqstroke
8915   \group_end:
8916 }
8917 { \@@_error:nmn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8918 }

```

If the key `name` has been used for the command `\SubMatrix`, we create a PGF node with that name for the submatrix (this node does not encompass the delimiters that we will put after).

```

8919   \str_if_empty:NF \l_@@_submatrix_name_str
8920   {
8921     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8922     \l_@@_x_initial_dim \l_@@_y_initial_dim
8923     \l_@@_x_final_dim \l_@@_y_final_dim
8924   }
8925   \group_end:

```

The group was for `\CT@arc@` (the color of the rules).

Now, we deal with the left delimiter. Of course, the environment `{pgfscope}` is for the `\pgftransformshift`.

```

8926   \begin { pgfscope }
8927   \pgftransformshift
8928   {
8929     \pgfpoint
8930     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8931     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8932   }
8933   \str_if_empty:NTF \l_@@_submatrix_name_str
8934   { \@@_node_left:nn #1 { } }
8935   { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8936   \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8937   \pgftransformshift
8938   {
8939     \pgfpoint
8940     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8941     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8942   }
8943   \str_if_empty:NTF \l_@@_submatrix_name_str
8944   { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8945   {
8946     \@@_node_right:nnnn #2
8947     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8948   }

```

Now, we deal with the key code of `\SubMatrix`. That key should contain a TikZ instruction and the nodes in that instruction will be relative to the current `\SubMatrix`. That's why we need a redefinition of `\pgfpointanchor`.

```

8949   \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8950   \flag_clear_new:N \l_@@_code_flag
8951   \l_@@_code_tl
8952   }

```

In the key code of the command `\SubMatrix` there may be TikZ instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $row-i$, $col-j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8953 \cs_set_eq:NN \@@_old_pgfpointanchor: \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument `#1` of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
8954 \cs_new:Npn \@@_pgfpointanchor:n #1
8955 { \exp_args:Ne \@@_old_pgfpointanchor: { \@@_pgfpointanchor_i:n { #1 } } }
```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`).

```
8956 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8957 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

8958 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8959 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8960 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
8961 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
8962 { \@@_pgfpointanchor_ii:n { #1 } }
8963 }
```

In the case where the name begins with `\tikz@pp@name`, we must retrieve the second `\tikz@pp@name`, that is to say to marker that we have added at the end (cf. `\@@_pgfpointanchor_i:n`).

```
8964 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8965 { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` or of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```
8966 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
8967 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
8968 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8969 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8970 { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retrieve the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
8971 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8972 }
```

The following function is for the case when the name contains an hyphen.

```
8973 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8974 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
8975 \@@_env:
8976 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8977 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8978 }
```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

8979 \tl_const:Nn \c_@@_integers_alist_tl
8980 {
8981   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8982   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8983   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8984   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8985 }

```

```

8986 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8987 {

```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form i - $|j$. That special form is the reason of the special form of the argument of `\pgfpointanchor` which arises with its command `\name_of_command` (see above).

In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

8988   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8989   {
8990     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8991     \@@_env: -
8992     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8993     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8994     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8995   }
8996   {
8997     \str_if_eq:eeTF { #1 } { last }
8998     {
8999       \flag_raise:N \l_@@_code_flag
9000       \@@_env: -
9001       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
9002       { \int_eval:n { \l_@@_last_i_tl + 1 } }
9003       { \int_eval:n { \l_@@_last_j_tl + 1 } }
9004     }
9005     { #1 }
9006   }
9007 }

```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

9008 \cs_new_protected:Npn \@@_node_left:nn #1 #2
9009 {
9010   \pgfnode
9011   { rectangle }
9012   { east }
9013   {
9014     \nullfont
9015     \c_math_toggle_token
9016     \@@_color:o \l_@@_delimiters_color_tl
9017     \left #1
9018     \vcenter
9019     {
9020       \nullfont
9021       \hrule \@height \l_tmpa_dim
9022       \@depth \c_zero_dim

```



```

9023         \@width \c_zero_dim
9024     }
9025     \right .
9026     \c_math_toggle_token
9027 }
9028 { #2 }
9029 { }
9030 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

9031 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
9032 {
9033     \pgfnode
9034     { rectangle }
9035     { west }
9036     {
9037         \nullfont
9038         \c_math_toggle_token
9039         \colorlet { current-color } { . }
9040         \@@_color:o \l_@@_delimiters_color_tl
9041         \left .
9042         \vcenter
9043         {
9044             \nullfont
9045             \hrule \@height \l_tmpa_dim
9046                 \@depth \c_zero_dim
9047                 \@width \c_zero_dim
9048         }
9049         \right #1
9050         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9051         ^ { \color { current-color } \smash { #4 } }
9052         \c_math_toggle_token
9053     }
9054     { #2 }
9055     { }
9056 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9057 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9058 {
9059     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
9060     \ignorespaces
9061 }
9062 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9063 {
9064     \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
9065     \ignorespaces
9066 }
9067 \keys_define:nn { nicematrix / Brace }
9068 {
9069     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9070     left-shorten .default:n = true ,
9071     left-shorten .value_forbidden:n = true ,

```

```

9072 right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9073 right-shorten .default:n = true ,
9074 right-shorten .value_forbidden:n = true ,
9075 shorten .meta:n = { left-shorten , right-shorten } ,
9076 shorten .value_forbidden:n = true ,
9077 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9078 yshift .value_required:n = true ,
9079 yshift .initial:n = \c_zero_dim ,
9080 color .tl_set:N = \l_tmpa_tl ,
9081 color .value_required:n = true ,
9082 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9083 }

```

#1 is the first cell of the rectangle (with the syntax $i-j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to under or over.

```

9084 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9085 {
9086   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9087   \@@_compute_i_j:nn { #1 } { #2 }
9088   \bool_lazy_or:nnTF
9089     { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
9090     { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
9091     {
9092       \str_if_eq:eeTF { #5 } { under }
9093       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9094       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9095     }
9096     {
9097       \tl_clear:N \l_tmpa_tl
9098       \keys_set:nn { nicematrix / Brace } { #4 }
9099       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9100       \pgfpicture
9101       \pgfrememberpicturepositiononpagetrue
9102       \pgf@relevantforpicturesizefalse
9103       \bool_if:NT \l_@@_brace_left_shorten_bool
9104         {
9105           \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9106           \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9107             {
9108               \cs_if_exist:cT
9109                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9110                 {
9111                   \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9112
9113                   \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
9114                     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9115                 }
9116             }
9117         }
9118       \bool_lazy_or:nnT
9119         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9120         { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
9121         {
9122           \@@_qpoint:n { col - \l_@@_first_j_tl }
9123           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9124         }
9125       \bool_if:NT \l_@@_brace_right_shorten_bool
9126         {
9127           \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9128           \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
9129             {

```

```

9130         \cs_if_exist:cT
9131         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9132         {
9133             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9134             \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
9135             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9136         }
9137     }
9138 }
9139 \bool_lazy_or:nnT
9140 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9141 { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
9142 {
9143     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9144     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9145 }
9146 \pgfset { inner~sep = \c_zero_dim }
9147 \str_if_eq:eeTF { #5 } { under }
9148 { \@@_underbrace_i:n { #3 } }
9149 { \@@_overbrace_i:n { #3 } }
9150 \endpgfpicture
9151 }
9152 \group_end:
9153 }

```

The argument is the text to put above the brace.

```

9154 \cs_new_protected:Npn \@@_overbrace_i:n #1
9155 {
9156     \@@_qpoint:n { row - \l_@@_first_i_tl }
9157     \pgftransformshift
9158     {
9159         \pgfpoint
9160         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9161         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9162     }
9163     \pgfnode
9164     { rectangle }
9165     { south }
9166     {
9167         \vtop
9168         {
9169             \group_begin:
9170             \everycr { }
9171             \halign
9172             {
9173                 \hfil ## \hfil \crcr
9174                 \bool_if:NTF \l_@@_tabular_bool
9175                 { \begin { tabular } { c } #1 \end { tabular } }
9176                 { $ \begin { array } { c } #1 \end { array } $ }
9177                 \cr
9178                 \c_math_toggle_token
9179                 \overbrace
9180                 {
9181                     \hbox_to_wd:nn
9182                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9183                     { }
9184                 }
9185                 \c_math_toggle_token
9186                 \cr
9187             }
9188             \group_end:
9189         }
9190     }
9191 }

```

```

9192     { }
9193 }

```

The argument is the text to put under the brace.

```

9194 \cs_new_protected:Npn \@_underbrace_i:n #1
9195 {
9196   \@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9197   \pgftransformshift
9198     {
9199     \pgfpoint
9200       { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9201       { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9202     }
9203   \pgfnode
9204     { rectangle }
9205     { north }
9206     {
9207     \group_begin:
9208     \everycr { }
9209     \vbox
9210       {
9211       \halign
9212         {
9213         \hfil ## \hfil \crcr
9214         \c_math_toggle_token
9215         \underbrace
9216           {
9217           \hbox_to_wd:nn
9218             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9219             { }
9220           }
9221         \c_math_toggle_token
9222         \cr
9223         \bool_if:NTF \l_@@_tabular_bool
9224           { \begin { tabular } { c } #1 \end { tabular } }
9225           { $ \begin { array } { c } #1 \end { array } $ }
9226         \cr
9227       }
9228     }
9229     \group_end:
9230   }
9231   { }
9232   { }
9233 }

```

35 The commands HBrace et VBrace

```

9234 \hook_gput_code:nnn { begindocument } { . }
9235 {
9236   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9237   {
9238     \tikzset
9239       {
9240       nicematrix / brace / .style =
9241         {
9242         decoration = { brace , raise = -0.15 em } ,
9243         decorate ,
9244       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9245         nicematrix / mirrored-brace / .style =
9246         {
9247             nicematrix / brace ,
9248             decoration = mirror ,
9249         }
9250     }
9251 }
9252 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9253 \keys_define:nn { nicematrix / Hbrace }
9254 {
9255     color .code:n = ,
9256     horizontal-label .code:n = ,
9257     horizontal-labels .code:n = ,
9258     shorten .code:n = ,
9259     shorten-start .code:n = ,
9260     shorten-end .code:n = ,
9261     unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
9262 }

```

Here we need an “fully expandable” command.

```

9263 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9264 {
9265     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9266     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9267     { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
9268 }

```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```

9269 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9270 {
9271     \int_compare:nNnTF { \c@iRow } < { 2 }
9272     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9273     \str_if_eq:nnTF { #2 } { * }
9274     {
9275         \NiceMatrixOptions { nullify-dots }
9276         \Ldots
9277         [
9278             line-style = nicematrix / brace ,
9279             #1 ,
9280             up =
9281             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9282         ]
9283     }
9284     {
9285         \Hdotsfor
9286         [
9287             line-style = nicematrix / brace ,
9288             #1 ,
9289             up =
9290             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9291         ]
9292         { #2 }
9293     }
9294 }
9295 {
9296     \str_if_eq:nnTF { #2 } { * }

```

```

9297     {
9298     \NiceMatrixOptions { nullify-dots }
9299     \Ldots
9300     [
9301     line-style = nicematrix / mirrored-brace ,
9302     #1 ,
9303     down =
9304     \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9305     ]
9306     }
9307     {
9308     \Hdotsfor
9309     [
9310     line-style = nicematrix / mirrored-brace ,
9311     #1 ,
9312     down =
9313     \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9314     ]
9315     { #2 }
9316     }
9317     }
9318     \keys_set:nn { nicematrix / Hbrace } { #1 }
9319     }

9320 \NewDocumentCommand { \@@_Vbrace } { 0 { } m m }
9321 {
9322 \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9323 { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9324 { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
9325 }

```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not).

```

9326 \cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
9327 {
9328 \int_compare:nNnTF { \c@jCol } < { 2 }
9329 {
9330 \str_if_eq:nnTF { #2 } { * }
9331 {
9332 \NiceMatrixOptions { nullify-dots }
9333 \Vdots
9334 [
9335 Vbrace ,
9336 line-style = nicematrix / mirrored-brace ,
9337 #1 ,
9338 down =
9339 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9340 ]
9341 }
9342 {
9343 \Vdotsfor
9344 [
9345 Vbrace ,
9346 line-style = nicematrix / mirrored-brace ,
9347 #1 ,
9348 down =
9349 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9350 ]
9351 { #2 }
9352 }
9353 }
9354 {
9355 \str_if_eq:nnTF { #2 } { * }
9356 {

```

```

9357     \NiceMatrixOptions { nullify-dots }
9358     \Vdots
9359     [
9360         Vbrace ,
9361         line-style = nicematrix / brace ,
9362         #1 ,
9363         up =
9364         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9365     ]
9366 }
9367 {
9368     \Vdotsfor
9369     [
9370         Vbrace ,
9371         line-style = nicematrix / brace ,
9372         #1 ,
9373         up =
9374         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9375     ]
9376     { #2 }
9377 }
9378 }
9379 \keys_set:nn { nicematrix / Hbrace } { #1 }
9380 }

```

36 The command TikzEveryCell

```

9381 \bool_new:N \l_@@_not_empty_bool
9382 \bool_new:N \l_@@_empty_bool
9383
9384 \keys_define:nn { nicematrix / TikzEveryCell }
9385 {
9386     not-empty .code:n =
9387         \bool_lazy_or:nnTF
9388         { \l_@@_in_code_after_bool }
9389         { \g_@@_create_cell_nodes_bool }
9390         { \bool_set_true:N \l_@@_not_empty_bool }
9391         { \@@_error:n { detection-of-empty-cells } } } ,
9392     not-empty .value_forbidden:n = true ,
9393     empty .code:n =
9394         \bool_lazy_or:nnTF
9395         { \l_@@_in_code_after_bool }
9396         { \g_@@_create_cell_nodes_bool }
9397         { \bool_set_true:N \l_@@_empty_bool }
9398         { \@@_error:n { detection-of-empty-cells } } } ,
9399     empty .value_forbidden:n = true ,
9400     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9401 }
9402
9403
9404 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9405 {
9406     \IfPackageLoadedTF { tikz }
9407     {
9408         \group_begin:
9409         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

9410     \tl_set:Nn \l_tmpa_tl { { #2 } }
9411     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9412     { \@@_for_a_block:nnnnn #1 }

```

```

9413     \@@_all_the_cells:
9414     \group_end:
9415     }
9416     { \@@_error:n { TikzEveryCell~without~tikz } }
9417 }
9418
9419 \tl_new:N \l_@@_i_tl
9420 \tl_new:N \l_@@_j_tl
9421
9422
9423 \cs_new_protected:Nn \@@_all_the_cells:
9424 {
9425     \int_step_inline:nn \c@iRow
9426     {
9427         \int_step_inline:nn \c@jCol
9428         {
9429             \cs_if_exist:cF { cell - ##1 - #####1 }
9430             {
9431                 \clist_if_in:Nef \l_@@_corners_cells_clist
9432                 { ##1 - #####1 }
9433                 {
9434                     \bool_set_false:N \l_tmpa_bool
9435                     \cs_if_exist:cTF
9436                     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
9437                     {
9438                         \bool_if:NF \l_@@_empty_bool
9439                         { \bool_set_true:N \l_tmpa_bool }
9440                     }
9441                     {
9442                         \bool_if:NF \l_@@_not_empty_bool
9443                         { \bool_set_true:N \l_tmpa_bool }
9444                     }
9445                     \bool_if:NT \l_tmpa_bool
9446                     {
9447                         \@@_block_tikz:onnnn
9448                         \l_tmpa_tl { ##1 } { #####1 } { ##1 } { #####1 }
9449                     }
9450                 }
9451             }
9452         }
9453     }
9454 }
9455
9456 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9457 {
9458     \bool_if:NF \l_@@_empty_bool
9459     {
9460         \@@_block_tikz:onnnn
9461         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9462     }
9463     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9464 }
9465
9466 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9467 {
9468     \int_step_inline:nnn { #1 } { #3 }
9469     {
9470         \int_step_inline:nnn { #2 } { #4 }
9471         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9472     }
9473 }

```


37 The command \ShowCellNames

```

9474 \NewDocumentCommand \@@_ShowCellNames { }
9475 {
9476   \bool_if:NT \l_@@_in_code_after_bool
9477   {
9478     \pgfpicture
9479     \pgfrememberpicturepositiononpagetrue
9480     \pgf@relevantforpicturesizefalse
9481     \pgfpathrectanglecorners
9482     { \@@_qpoint:n { 1 } }
9483     {
9484       \@@_qpoint:n
9485       { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
9486     }
9487     \pgfsetfillopacity { 0.75 }
9488     \pgfsetfillcolor { white }
9489     \pgfusepathqfill
9490     \endpgfpicture
9491   }
9492   \dim_gzero_new:N \g_@@_tmpc_dim
9493   \dim_gzero_new:N \g_@@_tmpd_dim
9494   \dim_gzero_new:N \g_@@_tmpe_dim
9495   \int_step_inline:nn { \c@iRow }
9496   {
9497     \bool_if:NTF \l_@@_in_code_after_bool
9498     {
9499       \pgfpicture
9500       \pgfrememberpicturepositiononpagetrue
9501       \pgf@relevantforpicturesizefalse
9502     }
9503     { \begin { pgfpicture } }
9504     \@@_qpoint:n { row - ##1 }
9505     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9506     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9507     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9508     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9509     \bool_if:NTF \l_@@_in_code_after_bool
9510     { \endpgfpicture }
9511     { \end { pgfpicture } }
9512     \int_step_inline:nn { \c@jCol }
9513     {
9514       \hbox_set:Nn \l_tmpa_box
9515       {
9516         \normalfont \Large \sffamily \bfseries
9517         \bool_if:NTF \l_@@_in_code_after_bool
9518         { \color { red } }
9519         { \color { red ! 50 } }
9520         ##1 - #####1
9521       }
9522       \bool_if:NTF \l_@@_in_code_after_bool
9523       {
9524         \pgfpicture
9525         \pgfrememberpicturepositiononpagetrue
9526         \pgf@relevantforpicturesizefalse
9527       }
9528       { \begin { pgfpicture } }
9529       \@@_qpoint:n { col - #####1 }
9530       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9531       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9532       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9533       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

9534     \bool_if:NTF \l_@@_in_code_after_bool
9535         { \endpgfpicture }
9536         { \end { pgfpicture } }
9537     \fp_set:Nn \l_tmpa_fp
9538         {
9539         \fp_min:nn
9540         {
9541             \fp_min:nn
9542             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9543             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9544         }
9545         { 1.0 }
9546     }
9547     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9548     \pgfpicture
9549     \pgfrememberpicturerepositiononpagetrue
9550     \pgf@relevantforpicturesizefalse
9551     \pgftransformshift
9552     {
9553         \pgfpoint
9554         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9555         { \dim_use:N \g_tmpa_dim }
9556     }
9557     \pgfnode
9558     { rectangle }
9559     { center }
9560     { \box_use:N \l_tmpa_box }
9561     { }
9562     { }
9563     \endpgfpicture
9564 }
9565 }
9566 }

```

38 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

9567 \bool_new:N \g_@@_footnotehyper_bool

```

The boolean `\g_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to `true` if the option `footnotehyper` is used.

```

9568 \bool_new:N \g_@@_footnote_bool
9569 \msg_new:nmmm { nicematrix } { Unknown~key~for~package }
9570 {
9571     You-have-used-the-key~' \l_keys_key_str '~when-loading~nicematrix~
9572     but-that-key-is-unknown. \\
9573     It-will-be-ignored. \\
9574     For~a~list~of~the~available~keys,~type-H~<return>.
9575 }
9576 {
9577     The~available~keys~are~(in~alphabetic~order):~
9578     footnote,~
9579     footnotehyper,~
9580     messages-for-Overleaf,~
9581     renew-dots~and~

```

```

9582   renew-matrix.
9583 }
9584 \keys_define:nn { nicematrix }
9585 {
9586   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9587   renew-dots .value_forbidden:n = true ,
9588   renew-matrix .code:n = \@@_renew_matrix: ,
9589   renew-matrix .value_forbidden:n = true ,
9590   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9591   footnote .bool_set:N = \g_@@_footnote_bool ,
9592   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9593   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9594 }
9595 \ProcessKeyOptions

9596 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9597 {
9598   You~can't~use~the~option~'footnote'~because~the~package~
9599   footnotehyper~has~already~been~loaded.~
9600   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9601   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9602   of~the~package~footnotehyper.\\
9603   The~package~footnote~won't~be~loaded.
9604 }
9605 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9606 {
9607   You~can't~use~the~option~'footnotehyper'~because~the~package~
9608   footnote~has~already~been~loaded.~
9609   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9610   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9611   of~the~package~footnote.\\
9612   The~package~footnotehyper~won't~be~loaded.
9613 }

9614 \bool_if:NT \g_@@_footnote_bool
9615 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9616   \IfClassLoadedTF { beamer }
9617     { \bool_set_false:N \g_@@_footnote_bool }
9618     {
9619       \IfPackageLoadedTF { footnotehyper }
9620         { \@@_error:n { footnote-with-footnotehyper~package } }
9621         { \usepackage { footnote } }
9622     }
9623 }
9624 \bool_if:NT \g_@@_footnotehyper_bool
9625 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9626   \IfClassLoadedTF { beamer }
9627     { \bool_set_false:N \g_@@_footnote_bool }
9628     {
9629       \IfPackageLoadedTF { footnote }
9630         { \@@_error:n { footnotehyper-with-footnote~package } }
9631         { \usepackage { footnotehyper } }
9632     }
9633   \bool_set_true:N \g_@@_footnote_bool
9634 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

39 About the package underscore

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```

9635 \bool_new:N \l_@@_underscore_loaded_bool
9636 \IfPackageLoadedT { underscore }
9637 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9638 \hook_gput_code:nnn { begindocument } { . }
9639 {
9640   \bool_if:NF \l_@@_underscore_loaded_bool
9641   {
9642     \IfPackageLoadedT { underscore }
9643     { \@@_error:n { underscore-after-nicematrix } }
9644   }
9645 }

```

40 Error messages of the package

```

9646 \str_const:Ne \c_@@_available_keys_str
9647 {
9648   \bool_if:nTF { ! \g_@@_messages_for_Overleaf_bool }
9649   { For-a-list-of-the-available-keys,type-H<return>. }
9650   { }
9651 }
9652 \seq_new:N \g_@@_types_of_matrix_seq
9653 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9654 {
9655   NiceMatrix ,
9656   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9657 }
9658 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9659 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9660 \cs_new_protected:Npn \@@_error_too_much_cols:
9661 {
9662   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9663   { \@@_fatal:nn { too-much-cols-for-array } }
9664   \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
9665   { \@@_fatal:n { too-much-cols-for-matrix } }
9666   \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
9667   { \@@_fatal:n { too-much-cols-for-matrix } }
9668   \bool_if:NF \l_@@_last_col_without_value_bool
9669   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9670 }

```

The following command must *not* be protected since it's used in an error message.

```

9671 \cs_new:Npn \@@_message_hdotsfor:
9672 {
9673   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9674   { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
9675     \token_to_str:N \Hbrace \ is~incorrect. }
9676 }

9677 \cs_new_protected:Npn \@@_Hline_in_cell:
9678 { \@@_fatal:n { Misuse~of~Hline } }

9679 \@@_msg_new:nn { Misuse~of~Hline }
9680 {
9681   Misuse~of~Hline. \\
9682   \token_to_str:N \Hline\ must~be~used~only~at~the~beginning~of~a~row.\\
9683   That~error~is~fatal.
9684 }

9685 \@@_msg_new:nn { hvlines,~rounded~corners~and~corners }
9686 {
9687   Incompatible~options.\\
9688   You~should~not~use~'hvlines',~'rounded~corners'~and~'corners'~at~the~same~time.\\
9689   The~output~will~not~be~reliable.
9690 }

9691 \@@_msg_new:nn { key~color~inside }
9692 {
9693   Key~deprecated.\\
9694   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9695   and~have~been~deprecated.\\
9696   You~won't~have~similar~message~till~the~end~of~the~document.
9697 }

9698 \@@_msg_new:nn { invalid~weight }
9699 {
9700   Unknown~key.\\
9701   The~key~' \l_keys_key_str '-of~your~column~X~is~unknown~and~will~be~ignored.
9702 }

9703 \@@_msg_new:nn { last~col~not~used }
9704 {
9705   Column~not~used.\\
9706   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9707   in~your~\@@_full_name_env: .~
9708   However,~you~can~go~on.
9709 }

9710 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9711 {
9712   Too~much~columns.\\
9713   In~the~row~ \int_eval:n { \c@iRow },~
9714   you~try~to~use~more~columns~
9715   than~allowed~by~your~ \@@_full_name_env: .
9716   \@@_message_hdotsfor: \
9717   The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
9718   (plus~the~exterior~columns).~This~error~is~fatal.
9719 }

9720 \@@_msg_new:nn { too~much~cols~for~matrix }
9721 {
9722   Too~much~columns.\\
9723   In~the~row~ \int_eval:n { \c@iRow } ,~
9724   you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
9725   \@@_message_hdotsfor: \
9726   Recall~that~the~maximal~number~of~columns~for~a~matrix~
9727   (excepted~the~potential~exterior~columns)~is~fixed~by~the~
9728   LaTeX~counter~'MaxMatrixCols'.~
9729   Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
9730   (use~ \token_to_str:N \setcounter \ to~change~that~value).~

```

```

9731     This~error~is~fatal.
9732 }

9733 \@@_msg_new:nn { too~much~cols~for~array }
9734 {
9735     Too~much~columns.\\
9736     In~the~row~ \int_eval:n { \c@iRow } ,~
9737     ~you~try~to~use~more~columns~than~allowed~by~your~
9738     \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
9739     \int_use:N \g_@@_static_num_of_col_int \
9740     \bool_if:nT
9741     { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
9742     { ~(plus~the~exterior~ones) }
9743     since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
9744     This~error~is~fatal.
9745 }

9746 \@@_msg_new:nn { columns~not~used }
9747 {
9748     Columns~not~used.\\
9749     The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '~
9750     It~announces~ \int_use:N \g_@@_static_num_of_col_int \
9751     columns~but~you~only~used~ \int_use:N \c@jCol .\\
9752     The~columns~you~did~not~used~won't~be~created.\\
9753     You~won't~have~similar~warning~till~the~end~of~the~document.
9754 }

9755 \@@_msg_new:nn { empty~preamble }
9756 {
9757     Empty~preamble.\\
9758     The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
9759     This~error~is~fatal.
9760 }

9761 \@@_msg_new:nn { in~first~col }
9762 {
9763     Erroneous~use.\\
9764     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9765     That~command~will~be~ignored.
9766 }

9767 \@@_msg_new:nn { in~last~col }
9768 {
9769     Erroneous~use.\\
9770     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9771     That~command~will~be~ignored.
9772 }

9773 \@@_msg_new:nn { in~first~row }
9774 {
9775     Erroneous~use.\\
9776     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9777     That~command~will~be~ignored.
9778 }

9779 \@@_msg_new:nn { in~last~row }
9780 {
9781     Erroneous~use.\\
9782     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9783     That~command~will~be~ignored.
9784 }

9785 \@@_msg_new:nn { TopRule~without~booktabs }
9786 {
9787     Erroneous~use.\\
9788     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9789     That~command~will~be~ignored.
9790 }

```

```

9791 \@@_msg_new:nn { TopRule~without~tikz }
9792 {
9793   Erroneous~use.\\
9794   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9795   That~command~will~be~ignored.
9796 }

9797 \@@_msg_new:nn { caption~outside~float }
9798 {
9799   Key~caption~forbidden.\\
9800   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9801   environment~(such~as~\{table\}).~This~key~will~be~ignored.
9802 }

9803 \@@_msg_new:nn { short~caption~without~caption }
9804 {
9805   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9806   However,~your~'short~caption'~will~be~used~as~'caption'.
9807 }

9808 \@@_msg_new:nn { double~closing~delimiter }
9809 {
9810   Double~delimiter.\\
9811   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9812   delimiter.~This~delimiter~will~be~ignored.
9813 }

9814 \@@_msg_new:nn { delimiter~after~opening }
9815 {
9816   Double~delimiter.\\
9817   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9818   delimiter.~That~delimiter~will~be~ignored.
9819 }

9820 \@@_msg_new:nn { bad~option~for~line~style }
9821 {
9822   Bad~line~style.\\
9823   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9824   is~'standard'.~That~key~will~be~ignored.
9825 }

9826 \@@_msg_new:nn { corners~with~no~cell~nodes }
9827 {
9828   Incompatible~keys.\\
9829   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9830   is~in~force.\\
9831   If~you~go~on,~that~key~will~be~ignored.
9832 }

9833 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9834 {
9835   Incompatible~keys.\\
9836   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9837   is~in~force.\\
9838   If~you~go~on,~those~extra~nodes~won't~be~created.
9839 }

9840 \@@_msg_new:nn { Identical~notes~in~caption }
9841 {
9842   Identical~tabular~notes.\\
9843   You~can't~put~several~notes~with~the~same~content~in~
9844   \token_to_str:N \caption \ (but~you~can~in~the~main~tabular).\\
9845   If~you~go~on,~the~output~will~probably~be~erroneous.
9846 }

9847 \@@_msg_new:nn { tabularnote~below~the~tabular }
9848 {
9849   \token_to_str:N \tabularnote \ forbidden\\
9850   You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~

```

```

9851 of~your~tabular~because~the~caption~will~be~composed~below~
9852 the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9853 key~'caption~above'~in~ \token_to_str:N \NiceMatrixOptions .\
9854 Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
9855 no~similar~error~will~raised~in~this~document.
9856 }

9857 \@@_msg_new:nn { Unknown~key~for~rules }
9858 {
9859   Unknown~key.\
9860   There~is~only~two~keys~available~here:~width~and~color.\
9861   Your~key~' \l_keys_key_str '~will~be~ignored.
9862 }

9863 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9864 {
9865   Unknown~key.\
9866   You~have~used~the~key~' \l_keys_key_str '~but~the~only~
9867 keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
9868 and~ \token_to_str:N \Vbrace \ are:~'color',~
9869 'horizontal~label(s)',~'shorten'~'shorten~end'~
9870 and~'shorten~start'.\
9871   That~error~is~fatal.
9872 }

9873 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9874 {
9875   Unknown~key.\
9876   There~is~only~two~keys~available~here:~
9877 'empty'~and~'not~empty'.\
9878   Your~key~' \l_keys_key_str '~will~be~ignored.
9879 }

9880 \@@_msg_new:nn { Unknown~key~for~rotate }
9881 {
9882   Unknown~key.\
9883   The~only~key~available~here~is~'c'.\
9884   Your~key~' \l_keys_key_str '~will~be~ignored.
9885 }

9886 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9887 {
9888   Unknown~key.\
9889   The~key~' \l_keys_key_str '~is~unknown~in~a~'custom~line'.~
9890   It~you~go~on,~you~will~probably~have~other~errors. \
9891   \c_@@_available_keys_str
9892 }
9893 {
9894   The~available~keys~are~(in~alphabetic~order):~
9895   ccommand,~
9896   color,~
9897   command,~
9898   dotted,~
9899   letter,~
9900   multiplicity,~
9901   sep~color,~
9902   tikz,~and~total~width.
9903 }

9904 \@@_msg_new:nnn { Unknown~key~for~xdots }
9905 {
9906   Unknown~key.\
9907   The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\
9908   \c_@@_available_keys_str
9909 }
9910 {
9911   The~available~keys~are~(in~alphabetic~order):~

```



```

9912     'color',~
9913     'horizontal(s)-labels',~
9914     'inter',~
9915     'line-style',~
9916     'radius',~
9917     'shorten',~
9918     'shorten-end'~and~'shorten-start'.
9919 }

9920 \@@_msg_new:nn { Unknown-key-for~rowcolors }
9921 {
9922     Unknown~key.\\
9923     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9924     (and~you~try~to~use~' \l_keys_key_str ')\\
9925     That~key~will~be~ignored.
9926 }

9927 \@@_msg_new:nn { label-without-caption }
9928 {
9929     You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
9930     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9931 }

9932 \@@_msg_new:nn { W-warning }
9933 {
9934     Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
9935     (row~ \int_use:N \c@iRow ).
9936 }

9937 \@@_msg_new:nn { Construct-too-large }
9938 {
9939     Construct~too~large.\\
9940     Your~command~ \token_to_str:N #1
9941     can't~be~drawn~because~your~matrix~is~too~small.\\
9942     That~command~will~be~ignored.
9943 }

9944 \@@_msg_new:nn { underscore-after-nicematrix }
9945 {
9946     Problem~with~'underscore'.\\
9947     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9948     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9949     ' \token_to_str:N \Cdots \token_to_str:N _
9950     \{ n \token_to_str:N \text \{ ~times \} \}'.
9951 }

9952 \@@_msg_new:nn { ampersand-in-light-syntax }
9953 {
9954     Ampersand~forbidden.\\
9955     You~can't~use~an~ampersand~( \token_to_str:N & )~to~separate~columns~because~
9956     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9957 }

9958 \@@_msg_new:nn { double-backslash-in-light-syntax }
9959 {
9960     Double~backslash~forbidden.\\
9961     You~can't~use~ \token_to_str:N \\
9962     ~to~separate~rows~because~the~key~'light-syntax'~
9963     is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
9964     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9965 }

9966 \@@_msg_new:nn { hlines-with-color }
9967 {
9968     Incompatible~keys.\\
9969     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9970     \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
9971     However,~you~can~put~several~commands~ \token_to_str:N \Block.\\

```

```

9972     Your~key~will~be~discarded.
9973 }
9974 \@@_msg_new:nn { bad~value~for~baseline }
9975 {
9976     Bad~value~for~baseline.\\
9977     The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
9978     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9979     \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
9980     the~form~'line-i'.\\
9981     A~value~of~1~will~be~used.
9982 }
9983 \@@_msg_new:nn { detection~of~empty~cells }
9984 {
9985     Problem~with~'not~empty'\\
9986     For~technical~reasons,~you~must~activate~
9987     'create~cell~nodes'~in~ \token_to_str:N \createBefore \
9988     in~order~to~use~the~key~' \l_keys_key_str '.\\
9989     That~key~will~be~ignored.
9990 }
9991 \@@_msg_new:nn { siunitx~not~loaded }
9992 {
9993     siunitx~not~loaded\\
9994     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9995     That~error~is~fatal.
9996 }
9997 \@@_msg_new:nn { Invalid~name }
9998 {
9999     Invalid~name.\\
10000     You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
10001     \SubMatrix \ of~your~ \@@_full_name_env: .\\
10002     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
10003     This~key~will~be~ignored.
10004 }
10005 \@@_msg_new:nn { Hbrace~not~allowed }
10006 {
10007     Command~not~allowed.\\
10008     You~can't~use~the~command~ \token_to_str:N #1
10009     because~you~have~not~loaded~
10010     \IfPackageLoadedTF { tikz }
10011     { the~TikZ~library~'decorations.pathreplacing'~.~Use~ }
10012     { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
10013     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
10014     That~command~will~be~ignored.
10015 }
10016 \@@_msg_new:nn { Vbrace~not~allowed }
10017 {
10018     Command~not~allowed.\\
10019     You~can't~use~the~command~ \token_to_str:N \Vbrace \
10020     because~you~have~not~loaded~TikZ~
10021     and~the~TikZ~library~'decorations.pathreplacing'.\\
10022     Use: ~\token_to_str:N \usepackage \{tikz\}~
10023     \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
10024     That~command~will~be~ignored.
10025 }
10026 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
10027 {
10028     Wrong~line.\\
10029     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
10030     \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
10031     number~is~not~valid.~It~will~be~ignored.
10032 }

```

```

10033 \@@_msg_new:nn { Impossible-delimiter }
10034 {
10035     Impossible~delimiter.\\
10036     It's~impossible~to~draw~the~#1~delimiter~of~your~
10037     \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
10038     in~that~column.
10039     \bool_if:NT \l_@@_submatrix_slim_bool
10040     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
10041     This~ \token_to_str:N \SubMatrix \ will~be~ignored.
10042 }
10043 \@@_msg_new:nnn { width-without-X-columns }
10044 {
10045     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
10046     the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
10047     That~key~will~be~ignored.
10048 }
10049 {
10050     This~message~is~the~message~'width~without~X~columns'~
10051     of~the~module~'nicematrix'.~
10052     The~experimented~users~can~disable~that~message~with~
10053     \token_to_str:N \msg_redirect_name:nnn .\\
10054 }
10055
10056 \@@_msg_new:nn { key-multiplicity-with-dotted }
10057 {
10058     Incompatible~keys. \\
10059     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
10060     in~a~'custom-line'.~They~are~incompatible. \\
10061     The~key~'multiplicity'~will~be~discarded.
10062 }
10063 \@@_msg_new:nn { empty-environment }
10064 {
10065     Empty~environment.\\
10066     Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
10067 }
10068 \@@_msg_new:nn { No-letter-and-no-command }
10069 {
10070     Erroneous~use.\\
10071     Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
10072     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10073     '~ccommand'~(to~draw~horizontal~rules).\\
10074     However,~you~can~go~on.
10075 }
10076 \@@_msg_new:nn { Forbidden-letter }
10077 {
10078     Forbidden~letter.\\
10079     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10080     It~will~be~ignored.\\
10081     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10082 }
10083 \@@_msg_new:nn { Several-letters }
10084 {
10085     Wrong~name.\\
10086     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10087     have~used~' \l_@@_letter_str ').\\
10088     It~will~be~ignored.
10089 }
10090 \@@_msg_new:nn { Delimiter-with-small }
10091 {
10092     Delimiter~forbidden.\\
10093     You~can't~put~a~delimiter~in~the~preamble~of~your~

```

```

10094 \@@_full_name_env: \
10095 because~the~key~'small'~is~in~force.\\
10096 This~error~is~fatal.
10097 }
10098 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10099 {
10100 Unknown~cell.\\
10101 Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
10102 the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
10103 can't~be~executed~because~a~cell~doesn't~exist.\\
10104 This~command~ \token_to_str:N \line \ will~be~ignored.
10105 }
10106 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10107 {
10108 Duplicate~name.\\
10109 The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
10110 in~this~ \@@_full_name_env: .\\
10111 This~key~will~be~ignored.\\
10112 \bool_if:NF \g_@@_messages_for_Overleaf_bool
10113 { For~a~list~of~the~names~already~used,~type-H~<return>. }
10114 }
10115 {
10116 The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
10117 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
10118 }
10119 \@@_msg_new:nn { r~or~l~with~preamble }
10120 {
10121 Erroneous~use.\\
10122 You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
10123 You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10124 your~ \@@_full_name_env: .\\
10125 This~key~will~be~ignored.
10126 }
10127 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10128 {
10129 Erroneous~use.\\
10130 You~can't~use~ \token_to_str:N \Hdotsfor \ in~an~exterior~column~of~
10131 the~array.~This~error~is~fatal.
10132 }
10133 \@@_msg_new:nn { bad~corner }
10134 {
10135 Bad~corner.\\
10136 #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10137 'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10138 This~specification~of~corner~will~be~ignored.
10139 }
10140 \@@_msg_new:nn { bad~border }
10141 {
10142 Bad~border.\\
10143 \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
10144 (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
10145 The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10146 also~use~the~key~'tikz'
10147 \IfPackageLoadedF { tikz }
10148 { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
10149 This~specification~of~border~will~be~ignored.
10150 }
10151 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10152 {
10153 TikZ~not~loaded.\\
10154 You~can't~use~ \token_to_str:N \TikzEveryCell \

```

```

10155     because-you-have-not-loaded-tikz.~
10156     This-command-will-be-ignored.
10157 }
10158 \@@_msg_new:nn { tikz-key-without-tikz }
10159 {
10160     TikZ-not-loaded.\\
10161     You-can't-use-the-key-'tikz'~for-the-command~' \token_to_str:N
10162     \Block '-because-you-have-not-loaded-tikz.~
10163     This-key-will-be-ignored.
10164 }
10165 \@@_msg_new:nn { Bad-argument-for-Block }
10166 {
10167     Bad-argument.\\
10168     The-first-mandatory-argument-of-\token_to_str:N \Block\ must-
10169     be-of-the-form-'i-j'~(or-completely-empty)~and-you-have-used:~
10170     '#1'. \\
10171     If-you-go-on,~the-\token_to_str:N \Block\ will-be-mono-cell~(as-if-
10172     the-argument-was-empty).
10173 }
10174 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
10175 {
10176     Erroneous-use.\\
10177     In~the~ \@@_full_name_env: ,~you-must-use-the-key~
10178     'last-col'~without~value.\\
10179     However,~you-can-go-on-for~this~time~
10180     (the-value-' \l_keys_value_tl '-will-be-ignored).
10181 }
10182 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10183 {
10184     Erroneous-use. \\
10185     In~\token_to_str:N \NiceMatrixOptions ,~you-must-use-the-key~
10186     'last-col'~without~value. \\
10187     However,~you-can-go-on-for~this~time~
10188     (the-value-' \l_keys_value_tl '-will-be-ignored).
10189 }
10190 \@@_msg_new:nn { Block-too-large-1 }
10191 {
10192     Block-too-large. \\
10193     You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is-
10194     too-small-for-that-block. \\
10195     This-block-and-maybe-others-will-be-ignored.
10196 }
10197 \@@_msg_new:nn { Block-too-large-2 }
10198 {
10199     Block-too-large. \\
10200     The-preamble-of-your~ \@@_full_name_env: \ announces~ \int_use:N
10201     \g_@@_static_num_of_col_int \
10202     columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
10203     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10204     (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
10205     This-block-and-maybe-others-will-be-ignored.
10206 }
10207 \@@_msg_new:nn { unknown-column-type }
10208 {
10209     Bad-column-type. \\
10210     The-column-type~'#1'~in~your~ \@@_full_name_env: \
10211     is~unknown. \\
10212     This-error-is-fatal.
10213 }
10214 \@@_msg_new:nn { unknown-column-type-multicolumn }

```

```

10215 {
10216   Bad-column~type. \\
10217   The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
10218   ~of~your~ \@@_full_name_env: \
10219   is~unknown. \\
10220   This~error~is~fatal.
10221 }
10222 \@@_msg_new:nn { unknown~column~type-S }
10223 {
10224   Bad-column~type. \\
10225   The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
10226   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10227   load~that~package. \\
10228   This~error~is~fatal.
10229 }
10230 \@@_msg_new:nn { unknown~column~type-S-multicolumn }
10231 {
10232   Bad-column~type. \\
10233   The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
10234   of~your~ \@@_full_name_env: \ is~unknown. \\
10235   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10236   load~that~package. \\
10237   This~error~is~fatal.
10238 }
10239 \@@_msg_new:nn { tabularnote~forbidden }
10240 {
10241   Forbidden~command. \\
10242   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10243   ~here.~This~command~is~available~only~in~
10244   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10245   the~argument~of~a~command~\token_to_str:N \caption \ included~
10246   in~an~environment~\{table\}. \\
10247   This~command~will~be~ignored.
10248 }
10249 \@@_msg_new:nn { borders~forbidden }
10250 {
10251   Forbidden~key.\\
10252   You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
10253   because~the~option~'rounded~corners'~
10254   is~in~force~with~a~non~zero~value.\\
10255   This~key~will~be~ignored.
10256 }
10257 \@@_msg_new:nn { bottomrule~without~booktabs }
10258 {
10259   booktabs~not~loaded.\\
10260   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10261   loaded~'booktabs'.\\
10262   This~key~will~be~ignored.
10263 }
10264 \@@_msg_new:nn { enumitem~not~loaded }
10265 {
10266   enumitem~not~loaded. \\
10267   You~can't~use~the~command~ \token_to_str:N \tabularnote \
10268   ~because~you~haven't~loaded~'enumitem'. \\
10269   All~the~commands~ \token_to_str:N \tabularnote \ will~be~
10270   ignored~in~the~document.
10271 }
10272 \@@_msg_new:nn { tikz~without~tikz }
10273 {
10274   Tikz~not~loaded. \\
10275   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~

```

```

10276     loaded.~If~you~go~on,~that~key~will~be~ignored.
10277 }
10278 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10279 {
10280     Tikz~not~loaded. \\
10281     You~have~used~the~key~'tikz'~in~the~definition~of~a~
10282     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10283     You~can~go~on~but~you~will~have~another~error~if~you~actually~
10284     use~that~custom~line.
10285 }
10286 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10287 {
10288     Tikz~not~loaded. \\
10289     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10290     command~' \token_to_str:N \Block ')~but~tikz~is~not~loaded.~
10291     That~key~will~be~ignored.
10292 }
10293 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10294 {
10295     Erroneous~use.\\
10296     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10297     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10298     The~key~'color'~will~be~discarded.
10299 }
10300 \@@_msg_new:nn { Wrong~last~row }
10301 {
10302     Wrong~number.\\
10303     You~have~used~'last~row= \int_use:N \l_@@_last_row_int '~but~your~
10304     \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
10305     If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
10306     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10307     without~value~(more~compilations~might~be~necessary).
10308 }
10309 \@@_msg_new:nn { Yet~in~env }
10310 {
10311     Nested~environments.\\
10312     Environments~of~nicematrix~can't~be~nested.\\
10313     This~error~is~fatal.
10314 }
10315 \@@_msg_new:nn { Outside~math~mode }
10316 {
10317     Outside~math~mode.\\
10318     The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
10319     (and~not~in~ \token_to_str:N \vcenter ).\\
10320     This~error~is~fatal.
10321 }
10322 \@@_msg_new:nn { One~letter~allowed }
10323 {
10324     Bad~name.\\
10325     The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
10326     you~have~used~' \l_keys_value_tl '.\\
10327     It~will~be~ignored.
10328 }
10329 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10330 {
10331     Environment~\{TabularNote\}~forbidden.\\
10332     You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
10333     but~*before*~the~ \token_to_str:N \CodeAfter . \\
10334     This~environment~\{TabularNote\}~will~be~ignored.
10335 }

```

```

10336 \@@_msg_new:nn { varwidth~not~loaded }
10337 {
10338   varwidth~not~loaded.\\
10339   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10340   loaded.\\
10341   Your~column~will~behave~like~'p'.
10342 }

10343 \@@_msg_new:nn { varwidth~not~loaded~in~X }
10344 {
10345   varwidth~not~loaded.\\
10346   You~can't~use~the~key~'V'~in~your~column~'X'~
10347   because~'varwidth'~is~not~loaded.\\
10348   It~will~be~ignored. \\
10349 }

10350 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10351 {
10352   Unknown~key.\\
10353   Your~key~' \l_keys_key_str ' ~is~unknown~for~a~rule.\\
10354   \c_@@_available_keys_str
10355 }
10356 {
10357   The~available~keys~are~(in~alphabetic~order):~
10358   color,~
10359   dotted,~
10360   multiplicity,~
10361   sep-color,~
10362   tikz,~and~total~width.
10363 }
10364

10365 \@@_msg_new:nnn { Unknown~key~for~Block }
10366 {
10367   Unknown~key. \\
10368   The~key~' \l_keys_key_str ' ~is~unknown~for~the~command~
10369   \token_to_str:N \Block . \\
10370   It~will~be~ignored. \\
10371   \c_@@_available_keys_str
10372 }
10373 {
10374   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10375   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10376   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10377   and~vlines.
10378 }

10379 \@@_msg_new:nnn { Unknown~key~for~Brace }
10380 {
10381   Unknown~key.\\
10382   The~key~' \l_keys_key_str ' ~is~unknown~for~the~commands~
10383   \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
10384   It~will~be~ignored. \\
10385   \c_@@_available_keys_str
10386 }
10387 {
10388   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10389   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10390   right~shorten)~and~yshift.
10391 }

10392 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10393 {
10394   Unknown~key.\\
10395   The~key~' \l_keys_key_str ' ~is~unknown.\\
10396   It~will~be~ignored. \\
10397   \c_@@_available_keys_str

```



```

10398 }
10399 {
10400   The-available-keys-are-(in~alphabetic~order):~
10401   delimiters/color,~
10402   rules~(with~the~subkeys~'color'~and~'width'),~
10403   sub-matrix~(several~subkeys)~
10404   and~xdots~(several~subkeys).~
10405   The~latter~is~for~the~command~ \token_to_str:N \line .
10406 }
10407 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10408 {
10409   Unknown~key.\\
10410   The~key~' \l_keys_key_str '~is~unknown.\\
10411   It~will~be~ignored. \\
10412   \c_@@_available_keys_str
10413 }
10414 {
10415   The-available-keys-are-(in~alphabetic~order):~
10416   create-cell-nodes,~
10417   delimiters/color~and~
10418   sub-matrix~(several~subkeys).
10419 }
10420 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10421 {
10422   Unknown~key.\\
10423   The~key~' \l_keys_key_str '~is~unknown.\\
10424   That~key~will~be~ignored. \\
10425   \c_@@_available_keys_str
10426 }
10427 {
10428   The-available-keys-are-(in~alphabetic~order):~
10429   'delimiters/color',~
10430   'extra-height',~
10431   'hlines',~
10432   'hvlines',~
10433   'left-xshift',~
10434   'name',~
10435   'right-xshift',~
10436   'rules'~(with~the~subkeys~'color'~and~'width'),~
10437   'slim',~
10438   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10439   and~'right-xshift').\\
10440 }
10441 \@@_msg_new:nnn { Unknown~key~for~notes }
10442 {
10443   Unknown~key.\\
10444   The~key~' \l_keys_key_str '~is~unknown.\\
10445   That~key~will~be~ignored. \\
10446   \c_@@_available_keys_str
10447 }
10448 {
10449   The-available-keys-are-(in~alphabetic~order):~
10450   bottomrule,~
10451   code-after,~
10452   code-before,~
10453   detect-duplicates,~
10454   enumitem-keys,~
10455   enumitem-keys-para,~
10456   para,~
10457   label-in-list,~
10458   label-in-tabular~and~
10459   style.
10460 }

```

```

10461 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10462 {
10463   Unknown~key.\\
10464   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10465   \token_to_str:N \RowStyle . \\
10466   That~key~will~be~ignored. \\
10467   \c_@@_available_keys_str
10468 }
10469 {
10470   The~available~keys~are~(in~alphabetic~order):~
10471   bold,~
10472   cell-space-top-limit,~
10473   cell-space-bottom-limit,~
10474   cell-space-limits,~
10475   color,~
10476   fill~(alias:~rowcolor),~
10477   nb-rows,~
10478   opacity~and~
10479   rounded-corners.
10480 }
10481 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10482 {
10483   Unknown~key.\\
10484   The~key~' \l_keys_key_str '~is~unknown~for~the~command~
10485   \token_to_str:N \NiceMatrixOptions . \\
10486   That~key~will~be~ignored. \\
10487   \c_@@_available_keys_str
10488 }
10489 {
10490   The~available~keys~are~(in~alphabetic~order):~
10491   &-in-blocks,~
10492   allow-duplicate-names,~
10493   ampersand-in-blocks,~
10494   caption-above,~
10495   cell-space-bottom-limit,~
10496   cell-space-limits,~
10497   cell-space-top-limit,~
10498   code-for-first-col,~
10499   code-for-first-row,~
10500   code-for-last-col,~
10501   code-for-last-row,~
10502   corners,~
10503   custom-key,~
10504   create-extra-nodes,~
10505   create-medium-nodes,~
10506   create-large-nodes,~
10507   custom-line,~
10508   delimiters~(several~subkeys),~
10509   end-of-row,~
10510   first-col,~
10511   first-row,~
10512   hlines,~
10513   hvlines,~
10514   hvlines-except-borders,~
10515   last-col,~
10516   last-row,~
10517   left-margin,~
10518   light-syntax,~
10519   light-syntax-expanded,~
10520   matrix/columns-type,~
10521   no-cell-nodes,~
10522   notes~(several~subkeys),~
10523   nullify-dots,~

```

```

10524 pgf-node-code,~
10525 renew-dots,~
10526 renew-matrix,~
10527 respect-arraystretch,~
10528 rounded-corners,~
10529 right-margin,~
10530 rules~(with~the~subkeys~'color'~and~'width'),~
10531 small,~
10532 sub-matrix~(several~subkeys),~
10533 vlines,~
10534 xdots~(several~subkeys).
10535 }

```

For `{NiceArray}`, the set of keys is the same as for `{NiceMatrix}` excepted that there is no `l` and `r`.

```

10536 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10537 {
10538   Unknown~key.\\
10539   The~key~' \l_keys_key_str ' ~is~unknown~for~the~environment~
10540   \{NiceArray\}. \\
10541   That~key~will~be~ignored. \\
10542   \c_@@_available_keys_str
10543 }
10544 {
10545   The~available~keys~are~(in~alphabetic~order):~
10546   &~in~blocks,~
10547   ampersand~in~blocks,~
10548   b,~
10549   baseline,~
10550   c,~
10551   cell-space-bottom-limit,~
10552   cell-space-limits,~
10553   cell-space-top-limit,~
10554   code-after,~
10555   code-for-first-col,~
10556   code-for-first-row,~
10557   code-for-last-col,~
10558   code-for-last-row,~
10559   columns-width,~
10560   corners,~
10561   create-extra-nodes,~
10562   create-medium-nodes,~
10563   create-large-nodes,~
10564   extra-left-margin,~
10565   extra-right-margin,~
10566   first-col,~
10567   first-row,~
10568   hlines,~
10569   hvlines,~
10570   hvlines-except-borders,~
10571   last-col,~
10572   last-row,~
10573   left-margin,~
10574   light-syntax,~
10575   light-syntax-expanded,~
10576   name,~
10577   no-cell-nodes,~
10578   nullify-dots,~
10579   pgf-node-code,~
10580   renew-dots,~
10581   respect-arraystretch,~
10582   right-margin,~
10583   rounded-corners,~
10584   rules~(with~the~subkeys~'color'~and~'width'),~

```

```

10585     small,~
10586     t,~
10587     vlines,~
10588     xdots/color,~
10589     xdots/shorten-start,~
10590     xdots/shorten-end,~
10591     xdots/shorten-and~
10592     xdots/line-style.
10593 }

```

This error message is used for the set of keys `nicematrix/NiceMatrix` and `nicematrix/pNiceArray` (but not by `nicematrix/NiceArray` because, for this set of keys, there is no `l` and `r`).

```

10594 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10595 {
10596   Unknown~key.\\
10597   The~key~' \l_keys_key_str '~is~unknown~for~the~
10598   \@@_full_name_env: . \\
10599   That~key~will~be~ignored. \\
10600   \c_@@_available_keys_str
10601 }
10602 {
10603   The~available~keys~are~(in~alphabetic~order):~
10604   &~in~blocks,~
10605   ampersand~in~blocks,~
10606   b,~
10607   baseline,~
10608   c,~
10609   cell~space~bottom~limit,~
10610   cell~space~limits,~
10611   cell~space~top~limit,~
10612   code~after,~
10613   code~for~first~col,~
10614   code~for~first~row,~
10615   code~for~last~col,~
10616   code~for~last~row,~
10617   columns~type,~
10618   columns~width,~
10619   corners,~
10620   create~extra~nodes,~
10621   create~medium~nodes,~
10622   create~large~nodes,~
10623   extra~left~margin,~
10624   extra~right~margin,~
10625   first~col,~
10626   first~row,~
10627   hlines,~
10628   hvlines,~
10629   hvlines~except~borders,~
10630   l,~
10631   last~col,~
10632   last~row,~
10633   left~margin,~
10634   light~syntax,~
10635   light~syntax~expanded,~
10636   name,~
10637   no~cell~nodes,~
10638   nullify~dots,~
10639   pgf~node~code,~
10640   r,~
10641   renew~dots,~
10642   respect~arraystretch,~
10643   right~margin,~
10644   rounded~corners,~
10645   rules~(with~the~subkeys~'color'~and~'width'),~

```

```

10646     small,~
10647     t,~
10648     vlines,~
10649     xdots/color,~
10650     xdots/shorten-start,~
10651     xdots/shorten-end,~
10652     xdots/shorten-and~
10653     xdots/line-style.
10654 }
10655 \@@_msg_new:nnn { Unknown-key-for-NiceTabular }
10656 {
10657   Unknown-key.\\
10658   The-key-' \l_keys_key_str 'is-unknown-for-the-environment~
10659   \{NiceTabular\}. \\
10660   That-key-will-be-ignored. \\
10661   \c_@@_available_keys_str
10662 }
10663 {
10664   The-available-keys-are-(in-alphabetic-order):~
10665   &-in-blocks,~
10666   ampersand-in-blocks,~
10667   b,~
10668   baseline,~
10669   c,~
10670   caption,~
10671   cell-space-bottom-limit,~
10672   cell-space-limits,~
10673   cell-space-top-limit,~
10674   code-after,~
10675   code-for-first-col,~
10676   code-for-first-row,~
10677   code-for-last-col,~
10678   code-for-last-row,~
10679   columns-width,~
10680   corners,~
10681   custom-line,~
10682   create-extra-nodes,~
10683   create-medium-nodes,~
10684   create-large-nodes,~
10685   extra-left-margin,~
10686   extra-right-margin,~
10687   first-col,~
10688   first-row,~
10689   hlines,~
10690   hvlines,~
10691   hvlines-except-borders,~
10692   label,~
10693   last-col,~
10694   last-row,~
10695   left-margin,~
10696   light-syntax,~
10697   light-syntax-expanded,~
10698   name,~
10699   no-cell-nodes,~
10700   notes~(several~subkeys),~
10701   nullify-dots,~
10702   pgf-node-code,~
10703   renew-dots,~
10704   respect-arraystretch,~
10705   right-margin,~
10706   rounded-corners,~
10707   rules~(with~the~subkeys~'color'~and~'width'),~
10708   short-caption,~

```

```

10709     t,~
10710     tabularnote,~
10711     vlines,~
10712     xdots/color,~
10713     xdots/shorten-start,~
10714     xdots/shorten-end,~
10715     xdots/shorten-and~
10716     xdots/line-style.
10717 }
10718 \@@_msg_new:nnn { Duplicate-name }
10719 {
10720     Duplicate-name.\\
10721     The-name-' \l_keys_value_tl 'is-already-used-and-you-shouldn't-use~
10722     the-same-environment-name-twice.~You-can-go-on,~but,~
10723     maybe,~you-will-have-incorrect-results-especially~
10724     if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
10725     message-again,~use-the-key-'allow-duplicate-names'-in~
10726     ' \token_to_str:N \NiceMatrixOptions '.\\
10727     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10728     { For-a-list-of-the-names-already-used,~type-H~<return>. }
10729 }
10730 {
10731     The-names-already-defined-in-this-document-are:~
10732     \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
10733 }
10734 \@@_msg_new:nn { Option-auto-for-columns-width }
10735 {
10736     Erroneous-use.\\
10737     You-can't-give-the-value-'auto'-to-the-key-'columns-width'~here.~
10738     That-key-will-be-ignored.
10739 }
10740 \@@_msg_new:nn { NiceTabularX-without-X }
10741 {
10742     NiceTabularX-without-X.\\
10743     You-should-not-use-\{NiceTabularX\}-without-X-columns.\\
10744     However,~you-can-go-on.
10745 }
10746 \@@_msg_new:nn { Preamble-forgotten }
10747 {
10748     Preamble-forgotten.\\
10749     You-have-probably-forgotten-the-preamble-of-your~
10750     \@@_full_name_env: . \\
10751     This-error-is-fatal.
10752 }
10753 \@@_msg_new:nn { Invalid-col-number }
10754 {
10755     Invalid-column-number.\\
10756     A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10757     specifies-a-column-which-is-outside-the-array.~It-will-be-ignored.
10758 }
10759 \@@_msg_new:nn { Invalid-row-number }
10760 {
10761     Invalid-row-number.\\
10762     A-color-instruction-in-the~ \token_to_str:N \CodeBefore \
10763     specifies-a-row-which-is-outside-the-array.~It-will-be-ignored.
10764 }
10765 \@@_define_com:NNN p ( )
10766 \@@_define_com:NNN b [ ]
10767 \@@_define_com:NNN v | |
10768 \@@_define_com:NNN V \l \l
10769 \@@_define_com:NNN B \{ \}

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	3
4	Parameters	9
5	The command <code>\tabularnote</code>	20
6	Command for creation of rectangle nodes	24
7	The options	25
8	Important code used by <code>{NiceArrayWithDelims}</code>	36
9	The <code>\CodeBefore</code>	50
10	The environment <code>{NiceArrayWithDelims}</code>	55
11	Construction of the preamble of the array	60
12	The redefinition of <code>\multicolumn</code>	76
13	The environment <code>{NiceMatrix}</code> and its variants	93
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	94
15	After the construction of the array	96
16	We draw the dotted lines	102
17	The actual instructions for drawing the dotted lines with Tikz	117
18	User commands available in the new environments	123
19	The command <code>\line</code> accessible in code-after	129
20	The command <code>\RowStyle</code>	130
21	Colors of cells, rows and columns	133
22	The vertical and horizontal rules	145
23	The empty corners	162
24	The environment <code>{NiceMatrixBlock}</code>	164
25	The extra nodes	165
26	The blocks	170
27	How to draw the dotted lines transparently	195
28	Automatic arrays	195
29	The redefinition of the command <code>\dotfill</code>	196
30	The command <code>\diagbox</code>	197

31	The keyword <code>\CodeAfter</code>	198
32	The delimiters in the preamble	199
33	The command <code>\SubMatrix</code>	200
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	209
35	The commands <code>HBrace</code> et <code>VBrace</code>	212
36	The command <code>TikzEveryCell</code>	215
37	The command <code>\ShowCellNames</code>	217
38	We process the options at package loading	218
39	About the package underscore	220
40	Error messages of the package	220